FlexGrid for WinForms

2022.04.19 更新

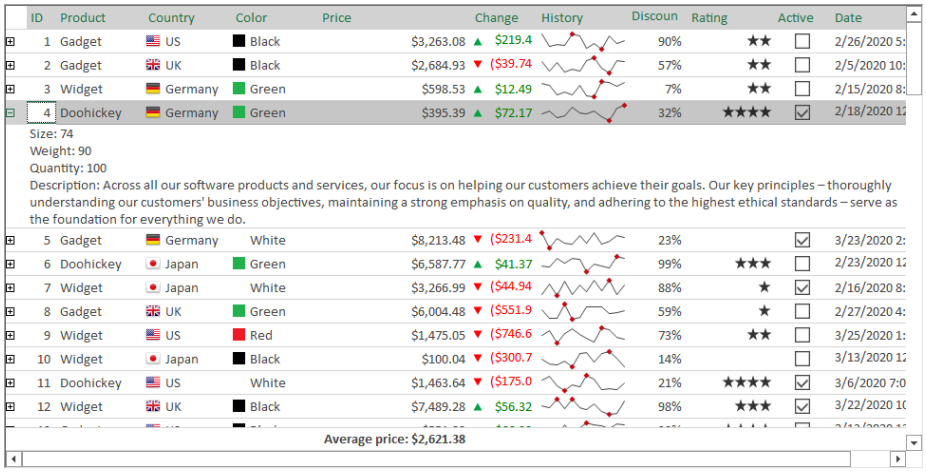
グレープシティ株式会社

FlexGrid for WinForms

目次

FlexGrid for WinForms

FlexGrid for WinForms は、最速クラスの市販データグリッドの 1 つで、大量のデータセットを他のどの .NET データグリッドより高速にレンダリングして表示します。セル編集、ソート、フィルタ処理、結合、グループ化などの基本機能や高度な機能を満載した強力なグリッドです。さらに、ツリーグリッドを複数列、集計値、小計値、行詳細と共に使用して、階層化データを効率的に表示します。それだけでなく、堅牢な API や広範な設計時サポートを備えており、エンドユーザーに使い慣れた Excel 的な操作性を提供できます。



FlexGrid コントロールは、ADO.NET データソースオブジェクトやカスタムビジネスオブジェクトなど、どのような .NET データソースとも連結します。また、非連結モードでも作業できるため、手動で行や列を追加してセル値を設定することができます。

FlexGrid の大きな強みの 1 つは、グリッド全体や個別のセルの外観をほとんどすべてカスタマイズできることです。FlexGrid は、標準的な書式文字列やセルのスタイル設定に加えて、OwnerDrawCell イベントを使用してセルの描画を完全に制御できることで、ほとんどの .NET データグリッドコンポーネントを凌駕しています。

リリースノート	**製品サンプル**
すべてのコントロールのバージョン別の更新については こちらを参照してください。	**ComponentOneControlPanel.exe** を使用して WinForms Edition をインストールする際にサンプルをインストールした場合、製品サンプルは、システムの \Documents\ComponentOne Samples\WinForms\vx.x.x\C1FlexGrid\CS にあります。
マニュアル	**ブログ**
初めての FlexGrid アプリケーションの作成	FlexGrid を使用して財務データを表示する
設計時の FlexGrid	FlexGrid のパフォーマンス: WinForms、WPF、UWP との比較
列とエディタの使用	FlexGrid のカスタムスタイルを作成する
グリッドデータのフィルタ処理	WinForms データグリッドに動的グループを追加する
FlexGrid での結合の有効化	WinForms データグリッドでレスポンシブな列を実装する
グループの作成	FlexGrid を使用して派生アコーディオンコントロールを作成する
ツリーグリッドの作成	ComponentOne FlexGrid ツリーに RadioButton を表示する
ビデオ	**デモサンプル**
FlexGrid の紹介	WinForms および WPF グリッドのデモ

メモ: ComponentOne FlexGrid for WinForms は .NET Framework および .NET 6 と互換性があります。

API リファレンス	
C1.Win.FlexGrid.4.5.2 アセンブリ	**C1.Win.FlexGrid.6 アセンブリ**

主な特長

コード不要の開発

ワンクリックでできるのに、まだコードを書きますか？ FlexGrid では、連結からスタイル設定に至るまで、豊富な設計時オプションが用意されています。これらはタスクメニューや各種エディタからアクセスでき、1 行のコードも記述することなく、さまざまな作業を行うことができます。

高度なセル編集

さまざまな組み込みエディタから選択することも、自分で作成することもできます。FlexGrid には、さまざまな組み込みセルエディタが用意されており、シンプルなテキスト編集、ドロップダウンリスト、コンボリスト、セルボタン、マスクなどから選択できます。それだけでなく、独自のカスタムセルエディタを作成することもでき、ほとんどすべての型のデータを受け取って表示するようにセルを変更できます。

連結モードまたは非連結モードでの作業

連結グリッドでも非連結グリッドでも、データをシームレスに挿入できます。FlexGrid では、ADO.NET や DataObjects for .NET. などの .NET データソースにグリッドを連結できます。また、非連結モードで作業して、FlexGrid でデータ自体を管理することもできます。

階層化データの提示

アプリケーション開発者やそのユーザーにとってベストな方法でデータを表示します。FlexGrid が階層化データソースに連結されると、マスターレコードをそれぞれ展開したり折りたたむことで、マスターレコードの詳細を子グリッド表示したり非表示にすることができます。その結果は、Microsoft Access に階層化データが表示される際に表示されるグリッドのタイプに近い「データツリー」になります。連結時、コントロールは下位のデータソースを検出し、子テーブルを表示するためのコントロールのインスタンスを追加して作成します。

データの集計と集計値の表示

グリッドデータを集計してすばやく概要を掴むことができます。FlexGrid では、小計行を追加し、指定した列に合計、平均、個数などの集計値を表示して、データを集計できます。

ツリー化

グリッドをツリーに変換することで、簡単に階層化データを表示できます。FlexGrid では、連結または非連結の階層化データを TreeView コントロールに似た ツリーグリッドで表現できます。ツリーグリッドは、わかりやすくアクセスも容易な構造をデータに与えます。

組み込みのデータフィルタ処理

ユーザーがデータを値または条件でフィルタ処理したり、そのどちらかを選択できるようにすることができます。FlexGrid には、高度な組み込みフィルタ処理として、列フィルタ、値フィルタ、条件フィルタが既に用意されています。それでは足りませんか？その場合は、機能豊富な FlexGrid API を使用して独自のカスタムフィルタを作成してください。

グループ化

グループ化は、プログラムで行うか、実行時に行います。FlexGrid ではどちらも可能です。FlexGrid では、コードからグループ化できるほか、FlexGridGroupPanel コントロールまたはコンテキストメニューを使用してユーザーが実行時にグループ化を行うオプションを提供することもできます。

クイック検索

グリッド全体を一気に検索して、数百万件ものレコードからエントリを見つけることができます。FlexGrid には FlexGridSearchPanel コントロールが用意されており、ユーザーが検索ボックスに入力すると、入力したテキストに一致するレコードをフィルタすると共に、検索結果を強調表示することもできます。

セルの結合

同じ値を持つセルを結合して、グリッドの見た目を整理できます。FlexGrid では、任意の方法で、同じ値を持つ連続したセル範囲を結合できます。無制限自動結合オプションや制限付き自動結合オプションを使用して、グリッドはどのセルを結合するかをインテリジェントに決定できます。

複数の形式の保存とロード

グリッドデータのバックアップや再ロードを一瞬で行うことができます。FlexGrid では、テキストファイル、Excel ファイル、XML など、好みの形式でグリッドを保存できます。これらの形式からコンテンツをロードすることもできます。それだけでなく、

DataReader オブジェクトを使用して、データベースからグリッドデータをロードすることもできます。

スパークラインの表示

グリッドを小さなチャートで生き生きと表現します。FlexGrid では、グリッド列にスパークラインを追加できます。これにより、トレンドや変動を 1 つのセルに簡単に表示し、グリッドデータをより便利で魅力的なものにします。

クリップボード操作の実行

Ctrl+C や Ctrl+V を使用して簡単にテキストを移動できます。FlexGrid では、好みのキーを使用して、グリッドデータやヘッダー上でクリップボードを操作できます。

列へのフィールド名の割り当て

列のインデックスを覚える必要はありません。列に名前を付けるだけです。グリッドがデータに連結されると、FlexGrid は自動的に列キーをフィールド名に割り当てます。または、コードから割り当てることができます。後から ColIndex(ColKey) 構文を使用して列を参照できます。この構文は、ユーザーがグリッド上の別の位置に列を移動した場合でも、目的の列を取得します。

データ注釈の使用

データ注釈でユーザーにヒントを提供できます。FlexGrid では、クラスや他のオブジェクトにメタデータタグの形式でデータ注釈を追加して、ユーザーにメッセージやヒントを表示します。

UI オートメーションとサポート

UI オートメーション（UIA）により、アクセス可能で機能豊富なクライアントアプリケーションを作成できます。FlexGrid コントロールは、TestStack.White などのフレームワークを使用した UI オートメーションをサポートしています。スクリーンリーダーのようなアプリケーション、ユーザーインタフェース要素を調査する UI テストコード、コードからのユーザー操作シミュレーションなども可能です。

> **メモ**: UI オートメーションは、C1FlexGrid のバージョン 4.5.2 および .NET framework 4.7 以上のアプリケーションターゲットフレームワークでのみ動作します。

特殊な描画効果の追加

特殊な描画効果を使用して、グリッドを一味違う魅力的な外観にします。FlexGrid では、グリッドセルで線、ビットマップ、アイコンなどの特殊な描画効果を使用できます。さらに、画像を拡大縮小したり、透過度を適用することもできます。

セル内の画像の表示

列内に画像を表示する必要がありませんか？それも大丈夫です。FlexGrid を使用すれば、データと一緒に画像を表示できます。グリッド列を画像リストに連結することもできます。これは、データソースから情報をグラフィカルに表示する簡単で効率的な方法です。

印刷機能の統合

1 つのステートメントでグリッドを印刷できます。FlexGrid では、用紙の向き、余白、ヘッダー/フッターテキストを制御でき、プリンタを設定したりプレビューを表示するためのダイアログボックスも提供されます。印刷イベントを使用して、各ページにページ区切りやカスタム要素を追加するなど、詳細な印刷オプションを実装することもできます。

スタイル設定

ビジュアルスタイルを使用することも、デザイナを使用したり独自コードを記述して独自のカスタムスタイルを作成することもできます。FlexGrid には、要件に合わせてグリッドの外観をカスタマイズするためのオプションが数多く用意されています。ヘッダーやアイコンから境界線や小計行に至るまで、FlexGrid のすべての要素をカスタマイズできます。

機能の比較

ここでは、複数のプラットフォームで利用可能なさまざまな FlexGrid の機能を比較すると共に、MS DataGridView との比較も示します。

トピック	コンテンツ
FlexGrid 間の比較	FlexGrid コントロールの ActiveX 版（VSFlexGrid コントロール）と .NET 版の違いを示し、グリッドを ActiveX から .NET に簡単に移行できるようにします。 • VSFlexGrid（ActiveX）と C1FlexGrid（.NET）
WinForms グリッド間の比較	ComponentOne WinForms Edition のグリッドと MS DataGridView のグリッドの機能を比較します。 • FlexGrid か True DBGrid か？ • ComponentOne のグリッドと MS DataGridView の比較

FlexGrid 間の比較

VSFlexGrid（ActiveX）と C1FlexGrid（.NET）

ComponentOne は、FlexGrid コントロールの ActiveX 版と .NET 版を提供しています。ActiveX 版は「**VSFlexGrid**」、.NET 版は「**C1FlexGrid**」という名前で公開されています。後から登場した .NET 版の C1FlexGrid は、既存の VSFlexGrid を移植するのではなく、まったく新しいグリッドコントロールとして作成されています。これは C# でーから記述されており、同じ設計原則に基づきながら、ActiveX コントロールよりモダンでクリーン、かつ強力な新しいオブジェクトモデルを備えています。

このセクションは、アプリケーションを ActiveX から .NET 版、つまり **C1FlexGrid** コントロールに移行したいと考えている既存の VSFlexGrid ユーザーのためのガイドです。手間のかからない円滑な移行とスムーズな学習を図ることができるように、.NET 版には **C1FlexGridClassic** コントロールが用意されています。このコントロールは、C1FlexGrid クラスから派生され、VSFlexGrid コントロールとほぼ同じオブジェクトモデルを提供します。新しいオブジェクトモデルの使用方法を正しく理解できるように、C1FlexGridClassic コントロールの製品サンプルが「**Classic（クラシック）**」として用意されています。**C1FlexGrid** クラスに基づいてカスタムグリッドコントロールを作成する際に、このサンプルを見本として使用することもできます。

> **メモ**: **ComponentOneControlPanel.exe** を使用して WinForms Edition をインストールする際にサンプルをインストールした場合、「**クラシック**」サンプルは、システムの \Documents\ComponentOne Samples\WinForms\v4.5.2\C1FlexGrid\CS にあります。

アプリケーションを新しく作成する場合は、新しい C1FlexGrid コントロールを使用することをお勧めします。ただし、既存のアプリケーションを ActiveX から .NET に移植する場合は、プログラミング作業が最小限で済むように C1FlexGridClassic コントロールを使用することをお勧めします。次の表に、VSFlexGrid と C1FlexGrid の主な違いを列挙します。

	VSFlexGrid（ActiveX）	C1FlexGrid（.NET）

行/列	1. **Rows** プロパティと **Cols** プロパティを使用して行数と列数を取得または設定します。
	2. VSFlexGrid は **TextMatrix** プロパティを使用して行と列を表します。
	
	たとえば、ActiveX では以下のように記述します。
	`Dim r%, c% c = 1`
	`For r = c1FlexGrid1.FixedRows To`
		1. C1FlexGrid では、**Rows** プロパティと **Cols** プロパティは、それぞれ行と列のコレクションを返します。これらのコレクションには、各コレクションの要素数と固定要素数を返す読み取り/書き込みプロパティがあります。
		2. C1FlexGrid はインデクサを使用して行と列を表します。
		
		たとえば、.NET では以下のように記述します。

	c1FlexGrid1.Rows - 1 Debug.Print c1FlexGrid1.TextMatrix(r,c) Next	Dim c As Integer = 1 For r = c1FlexGrid1.Rows.Fixed To c1FlexGrid1.Rows.Count - 1 Debug.Print(c1FlexGrid1(r, c)) Next
セル/セル範囲	**Cell** プロパティは、VSFlexGrid オブジェクトモデルの最も強力な要素の 1 つです。これを使用すると、任意のセルまたはセル範囲の任意のプロパティを 1 つのコマンドで取得または設定できます。単一のプロパティを使用するということは、バリアントを使用するということです。これにより、コードの作成中に何か間違いがあっても、細かな問題の多くをコンパイラが捕捉できなくなります。 たとえば、ActiveX では以下のように記述します。 flex.Cell(flexcpPicture, 5, 5, 10, 10) = theImage	C1FlexGrid では、Cell プロパティの代わりに **CellRange** オブジェクトが、セル範囲にアクセスするためのタイプセーフなプロパティとメソッドを公開しています。したがって、theImage 変数に画像ではなく文字列が含まれていると、実行時エラーではなくコンパイラエラーが発生します。また、各プロパティの型が既知なので、コードを記述する際にコマンド補完が働きます。 たとえば、.NET では以下のように記述します。 Dim rg As CellRange rg = c1FlexGrid1.GetCellRange(5,5,10,10) rg.Image = theImage
列の型指定	ActiveX 版では、**ColDataType** プロパティを使用して、各列に含まれるデータの型を設定できます。この情報は、データまたは数値が含まれている列をソートする際に主に使用されます。	.NET 版では、列が保持するデータの型は **Cols[i].DataType** プロパティによって決定されます。デフォルトで、すべての列の **DataType** プロパティは「object」に設定されています。つまり、任意の列に任意の型のデータを格納できます。データ型を特定の型に設定することは可能です。ただし、グリッドは、グリッド内に格納されたデータを適切な型に強制変換しようとします。 たとえば、.NET では以下のように記述します。 c1FlexGrid1.Cols(2).DataType = GetType(Integer) ' 値を 12 に設定します c1FlexGrid1(1, 2) = "12" ' 文字列「hello」は整数に変換できません ' GridError イベントが発生します ' 元の値が維持されます c1FlexGrid1(2, 2) = "hello"
スタイル	VSFlexGrid では、**Cell** プロパティを使用して個々のセルやセル範囲の外観をカスタマイズします。 たとえば、ActiveX では以下のように記述します。 //2 番目の行の背景色を設定します c1FlexGrid1.Cell(flexcpBackColor, 2, 0, 2, c1FlexGrid1.Cols-1) = vbRed **デメリット**:	C1FlexGrid では、セルの外観は **CellStyle** オブジェクトを使用してカスタマイズします。 たとえば、.NET では以下のように記述します。 //セルスタイルを作成します Dim redStyle As CellStyle = c1FlexGrid1.Styles .Add("Red") redStyle.BackColor = Color.Red //2 番目の行の背景色を設定します c1FlexGrid1.Rows(2).Style =

	すべての赤色セルの外観を変更するには、すべてのスタイルをクリアしてやり直すか、赤色セルをスキャンして外観を変更する必要があります。	`redStyle`
		メリット:
		このアプローチの主な利点は、新しいスタイルは、変更したり新しい範囲に割り当てることができるオブジェクトだということです。たとえば、赤色セルの前景色とテキストフォントを変更する場合は、以下のように記述します。
		`c1FlexGrid1.Styles("Red").ForeColor = Color.White`
		`c1FlexGrid1.Styles("Red").Font = new Font("Arial", 9)`

	VSFlexGrid コントロールには、グリッドの表示方法に影響する多くのプロパティがありました（BackColor、BackColorAlternate、BackColorBkg、BackColorFixed、BackColorFrozen、BackColorSel など）。	C1FlexGrid コントロールでは、これらのプロパティはすべて CellStyle オブジェクトのコレクションに置き換えられます。したがって、**Styles.Fixed.BackColor** または **Styles.Highlight.ForeColor** のような記述が可能です。これにより、オブジェクトモデルがよりシンプルで一貫性が高く、さらに強力になります。組み込みスタイルを変更したり、カスタムスタイルを定義して行、列、または任意のセル範囲に割り当てることができます。

WinForms グリッド間の比較

FlexGrid か True DBGrid か？

ComponentOne は WinForms Edition で FlexGrid と True DBGrid の 2 つのグリッドコンポーネントを提供しています。どちらも表形式のデータを参照、編集、追加、削除、操作するための堅牢で使いやすいグリッドコントロールですが、それぞれに特長と独自の機能があります。

どちらの ComponentOne WinForms Edition グリッドも連結モードおよび非連結モードで使用できますが、連結モードでの動作は C1FlexGrid の方が優れています。C1FlexGrid を使用すると、ツリーをカスタマイズしたり、セル結合機能を利用することができます。FlexGrid は、カスタマイズされたグリッドの派生にも適しています。

.NET 5 以上での開発や移行をお考えの場合は、C1FlexGrid の使用をお勧めします。C1TrueDBGrid は .NET 5 でも引き続きサポートされますが、C1FlexGrid のように機能が強化されたり新しい機能が追加されることはありません。

ComponentOne のグリッドと MS DataGridView の比較

このセクションでは、2 つの ComponentOne WinForms Edition グリッドと、WinForms プラットフォームにある標準の MS DataGridView を簡単に比較します。わかりやすいように、機能をさまざまなカテゴリに大まかに分類してあります。右側のペインでカテゴリをクリックするか、下にスクロールすることで、特定の機能を利用できるかどうかを確認できます。

データ連結

機能	FlexGrid	TrueDBGrid	MS DataGridView
データソースの連結	✓	✓	✓
データソースと階層化データリレーションの連結	カスタムコードを使用	✓	
非連結データストレージと操作	✓	✓	✓

データプレゼンテーション

機能	FlexGrid	TrueDBGrid	MS DataGridView
階層化されたスタイル	カスタムコードを使用	✓	
TreeView のようなスタイル	✓	✓	

複数行データビュー		✓	
グループ化	✓	✓	
組み込みのドラッグアンドドロップ	✓	✓	
インタラクティブな左右分割と上下分割		✓	
マスターグリッド内での子グリッドのサポート	✓	✓	
ドロップダウンオブジェクトのサポート	✓	✓	
ドロップダウン複数列オブジェクトのサポート	✓	✓	
ドロップダウン複数列連結可能およびソート可能オブジェクトのサポート		✓	

データ交換

機能	FlexGrid	TrueDBGrid	MS DataGridView
データのエクスポート（区切りテキスト、XLS、XLSX）	✓	✓	
他の形式へのデータのエクスポート (PDF、HTML、RTF、JPG など)		✓	
Excel ファイルからのデータのロード	✓	✓	
グリッドデータの拡張印刷および印刷プレビューのサポート	✓	✓	

セルの操作

機能	FlexGrid	TrueDBGrid	MS DataGridView
セル内オブジェクト	✓	✓	✓
カスタムエディタを使用した拡張セル編集	✓	✓	
セルや行の結合	✓	✓	
セル結合のカスタマイズ	✓		
列と行のドラッグアンドドロップ	✓	✓	
自動セルサイズ変更	✓	✓	✓
スクロールしない固定列	✓	✓	✓
Excel スタイルのセル選択	✓	✓	
各セルレンダリングのカスタマイズ	✓		
セルのズーム	✓	✓	
実行時の CellTip	✓	✓	
セル範囲を使用したデータ操作	✓		

レイアウトとスタイル設定

機能	FlexGrid	TrueDBGrid	MS DataGridView
ビジュアルスタイルのサポート	✓	✓	✓
38 種類の装飾テーマの動的サポート	✓	✓	
Office 2007 および 2010 スタイルの設定	✓	✓	
1 行おきに行の色を変える	✓	✓	✓
カスタマイズ可能なセル境界線スタイル	✓	✓	
特殊な描画効果の追加	✓	✓	
データ依存の表示	✓	✓	

入力とナビゲーション

機能	FlexGrid	TrueDBGrid	MS DataGridView
入力マスク	✓	✓	✓
データ入力の簡素化	✓	✓	
自動データ変換	✓	✓	
スペルチェックのサポート	✓	✓	
カスタマイズ可能なキーボードナビゲーションとキー動作	✓	✓	
右から左のナビゲーション	✓	✓	✓
タッチサポート	✓	✓	✓
クリップボードサポート	✓	✓	✓
豊富なスクロール機能	✓	✓	

データ操作

機能	FlexGrid	TrueDBGrid	MS DataGridView
ソート	✓	✓	✓
複数列のソート	✓		
組み込みのデータフィルタ処理	✓	✓	
拡張フィルタ処理と条件付きフィルタ処理	✓	✓	
カスタマイズ可能なフィルタフォーム	✓		
追加のフィルタバー行		✓	
多言語フィルタ処理	✓	✓	
AutoSearch	✓		
範囲集計値	✓	✓	

ローカライズ

機能	FlexGrid	TrueDBGrid	MS DataGridView
右から左のサポート	✓	✓	✓
.NET ローカライズのサポート	✓	✓	✓
地域設定のサポート	✓	✓	✓

その他の機能

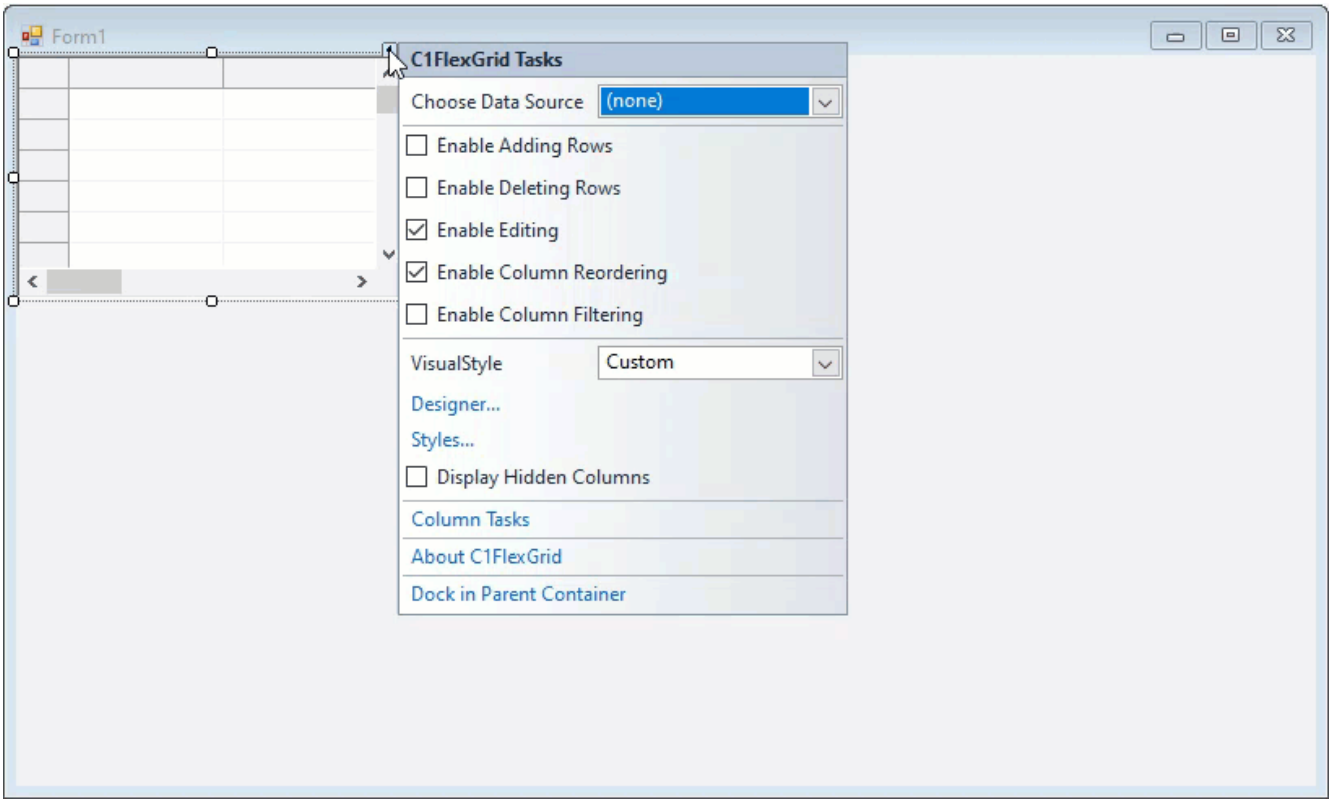
機能	FlexGrid	TrueDBGrid	MS DataGridView
連結および非連結モードで大量のデータを表示する場合のパフォーマンスの最適化	✓	✓	
ジャストインタイムデータロード	✓	✓	✓
C1DataSource によるサーバー側データ仮想化	✓	✓	✓
C1DataSource を使用した自動ルックアップ列	✓		✓
設計時の拡張サポート	✓	✓	
アセンブリサイズ	1508 K	2108 K	System.Windows.Forms の一部

クイックスタート

このクイックスタートでは、FlexGrid コントロールを使用して、シンプルなグリッドアプリケーションを作成する手順を紹介します。以下の手順に従って開始してください。

.NET framework

このセクションでは、外部データソースに連結して、.NET Framework で FlexGrid アプリケーションを作成する方法を具体的に示します。また、.NET 6 タブに示すように、グリッドにデータを供給するクラスを作成することもできます。同様に、以下の手順を使用して、.NET 6 で FlexGrid をデータソースに連結できます。ただし、.NET 6 の場合は、設計時の手順が一部異なります。



アプリケーションの設定

1. 新しい Windows Forms アプリケーション(.NET Framework)を作成します。
2. プロジェクトを構成し、Framework プロパティを設定します。
3. FlexGrid コントロールを Visual Studio ツールボックスからフォームにドラッグアンドドロップします。
   注意:設計時には空のグリッドがフォームに追加されます。

データソースに FlexGrid を連結

データを FlexGrid コントロールに連結する方法には、設計時に連結を選択する方法と、実行時にコードを通して行う方法の 2 つがあります。

設計時の連結

1. デザインビューで、FlexGrid コントロールを選択し、スマートタグをクリックして**C1FlexGrid タスク**メニューを開きます。
2. **[データソースの選択]**ドロップダウンボタンをクリックし、**[プロジェクトデータソースの追加]**オプションを選択して**データソース構成ウィザード**を開きます。
3. **[データソースの種類の選択]** ページで、**[データベース]**を選択し、**[次へ]**をクリックします。
4. **[データベースモデルの選択]**ページで、**[データセット]**を選択し、**[次へ]**をクリックします。
5. **[データ接続の選択]**ページで、**[新しい接続]** をクリックして **[接続の追加]** ダイアログを開きます。

6. **[データソース]**フィールドに対して、**[選択]**ボタンをクリックして **[データソースの変更]**ダイアログを開きます。
7. **[Microsoft Access データベースファイル]**を選択し、**[OK]** をクリックして[接続の追加]ダイアログに戻ります。
8. **[データベースファイル名]**フィールドに対して、**[参照]**をクリックして、データベースファイルに移動します。データベースファイルへの接続に必要な場合は、**ユーザー名**と**パスワード**を指定します。この例では、デフォルトで次の場所にある **C1NWind.mdb** ファイルを使用します。
\Documents\ComponentOne Samples\Common
9. **[接続のテスト]**をクリックしてデータベースまたはサーバーに正しく接続されていることを確認し、**[OK]**をクリックします。
10. **[OK]**をクリックして[接続の追加]ダイアログボックスを閉じます。
11. **[次へ]**をクリックして続行します。データファイルをプロジェクトに追加し、接続文字列を修正するかどうかを確認するダイアログボックスが表示されます。要件に従って適切なオプションを選択します。
12. **[次の名前で接続を保存する]**ボックスをオンにし、名前を入力して、接続文字列をアプリケーション構成ファイルに保存します。
13. **[次へ]**をクリックして**[データベースオブジェクトの選択]**ページに切り替えます。
14. **[テーブル]**ノードから、たとえば **Products** テーブルを選択し、**[終了]**をクリックします。
15. 以上の手順で、プロジェクトにデータセットと接続文字列が追加されます。また、Visual Studio は、データセットを設定するための以下のコードを自動的に作成します。

C#

```
// 次のサンプルコードは、データを「c1NWindDataSet.Products」テーブルにロードします。 必要に応じて、移動または削除できます。
this.productsTableAdapter.Fill(this.c1NWindDataSet.Products);
```

VB.NET

```
' 次のサンプルコードは、データを「c1NWindDataSet.Products」テーブルにロードします。 必要に応じて、移動または削除できます。
Me.ProductsTableAdapter.Fill(Me.c1NWindDataSet.Products)
```

実行時の連結

コードを通してグリッドを連結するには、まずデータベース接続文字列を作成する必要があります。次に、データアダプタのオブジェクト(この場合は、OleDbDataAdapter)を使用して、データテーブルから製品を取得するクエリーを作成し、そのデータを C1FlexGrid クラスの DataSource プロパティに割り当てられるデータテーブルに挿入します。

C#

```
// グリッドをデータソースに連結します
string conn = GetConnectionString();
OleDbDataAdapter da = new OleDbDataAdapter("select * from products", conn);
DataTable dt = new DataTable("Products");
da.Fill(dt);
c1FlexGrid1.DataSource = dt;
```

VB.NET

```
' グリッドをデータソースに連結します
Dim conn As String = GetConnectionString()
Dim da As OleDbDataAdapter = New OleDbDataAdapter("select * from products", conn)
Dim dt As DataTable = New DataTable("Products")
da.Fill(dt)
c1FlexGrid1.DataSource = dt
```

上記のサンプルコードは、**GetConnectionString** という名前のカスタムメソッドを使用してデータベースとの接続文字列を作成します。

C#

```
static string GetConnectionString()
```

```
{
    string path = Environment.GetFolderPath(Environment.SpecialFolder.Personal) +
@"\ComponentOne Samples\Common";
    string conn = @"provider=microsoft.jet.oledb.4.0;data source={0}\c1nwind.mdb;";
    return string.Format(conn, path);
}
```

VB.NET

```
Private Shared Function GetConnectionString() As String
    Dim path = Environment.GetFolderPath(Environment.SpecialFolder.Personal) &
"\ComponentOne Samples\Common"
    Dim conn = "provider=microsoft.jet.oledb.4.0;data source={0}\c1nwind.mdb;"
    Return String.Format(conn, path)
End Function
```

FlexGrid コントロールの構成

このセクションでは、設計時および実行時に、いくつかの基本設定を使用してグリッドを構成する手順を説明します。これらの設定と機能については、以下のトピックで詳細に説明します。

設計時のグリッドの構成

- スマートタグをクリックして **C1FlexGrid タスクメニュー**を開きます。
- **[スタイル]**オプションをクリックして、**C1FlexGrid スタイルエディタ**を開きます。
- **[組み込みスタイル]**ペインから**[固定]**を選択し、**Backcolor**、**Font**、**ForeColor** などの設定をカスタマイズして、**[OK]**をクリックします。
- グリッドをフォームに合わせるには、**[親コンテナにドッキングする]**オプションをクリックします。
- 列のカスタマイズとしては、たとえば **Unit Price** 列をクリックして、**C1FlexGrid 列タスクメニュー**を開きます。
- **[書式]**フィールドの横の省略符をクリックして、**[書式文字列]**ダイアログを開きます。
- **[書式の種類]**として**[通貨]**を選択し、**[OK]**をクリックします。

実行時のグリッドの構成

以下のコードを追加して、実行時にグリッドとその列を構成します。

C#

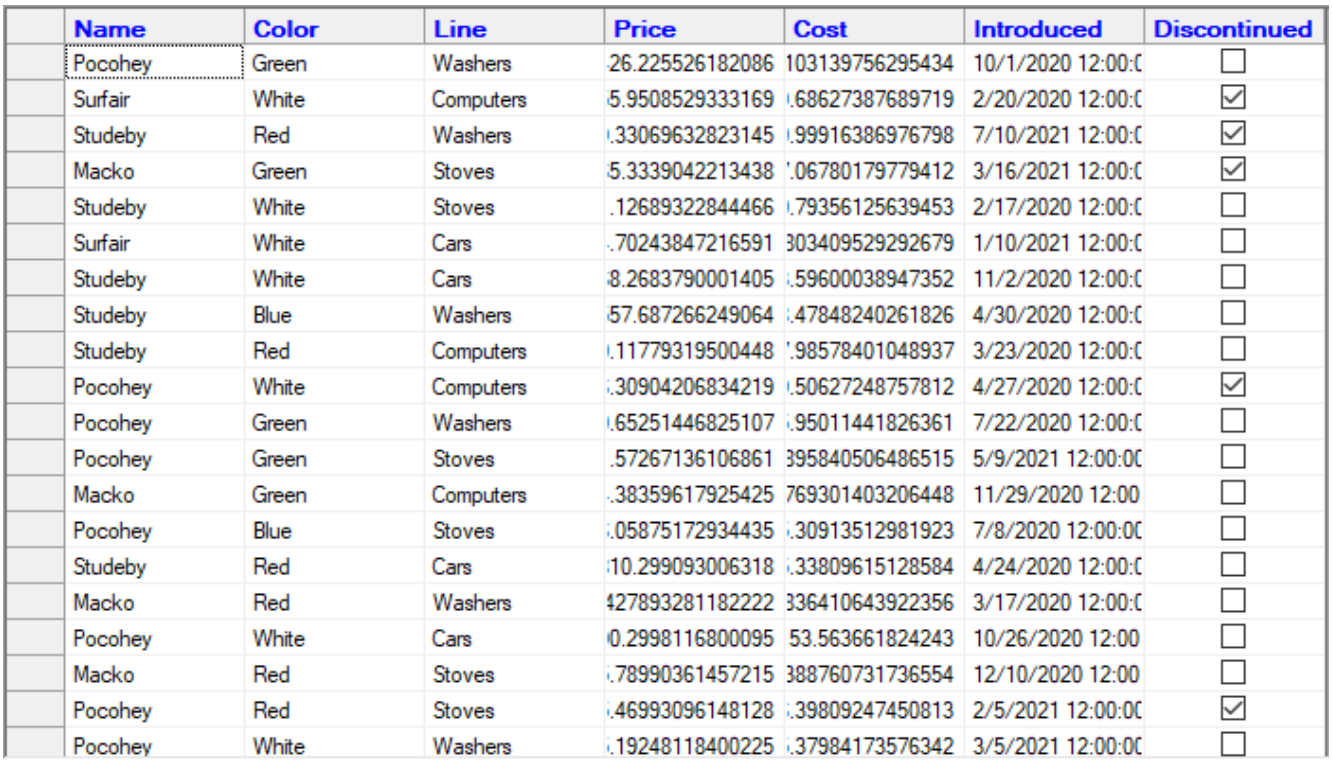
```
c1FlexGrid1.Styles.Fixed.ForeColor = Color.Blue;
c1FlexGrid1.Styles.Fixed.Font = new Font("Microsoft Sans serif", 9, FontStyle.Bold);
c1FlexGrid1.Dock = DockStyle.Fill;
c1FlexGrid1.Cols[6].Format = "c";
```

VB.NET

```
c1FlexGrid1.Styles.Fixed.ForeColor = Color.Blue
c1FlexGrid1.Styles.Fixed.Font = New Font("Microsoft Sans serif", 9, FontStyle.Bold)
c1FlexGrid1.Dock = DockStyle.Fill
c1FlexGrid1.Cols(6).Format = "c"
```

.NET 6

このセクションでは、カスタムクラスからデータを挿入して、.NET 6 で FlexGrid アプリケーションを作成する方法を具体的に示します。また、.NET Framework タブに示すように、外部データソースを使用してグリッドにデータを供給することもできます。ただし、.NET 6 の場合は、前述のタブで示した設計時の手順が一部異なります。同様に、以下の手順を使用して、.NET Framework で FlexGrid を内部データソースに連結できます。



	Name	Color	Line	Price	Cost	Introduced	Discontinued
	Pocohey	Green	Washers	26.225526182086	103139756295434	10/1/2020 12:00:0	☐
	Surfair	White	Computers	5.9508529333169	.68627387689719	2/20/2020 12:00:0	☑
	Studeby	Red	Washers	.33069632823145	.99916386976798	7/10/2021 12:00:0	☑
	Macko	Green	Stoves	5.3339042213438	.06780179779412	3/16/2021 12:00:0	☑
	Studeby	White	Stoves	.12689322844466	.79356125639453	2/17/2020 12:00:0	☐
	Surfair	White	Cars	.70243847216591	303409529292679	1/10/2021 12:00:0	☐
	Studeby	White	Cars	8.2683790001405	.59600038947352	11/2/2020 12:00:0	☐
	Studeby	Blue	Washers	57.687266249064	.47848240261826	4/30/2020 12:00:0	☐
	Studeby	Red	Computers	.11779319500448	.98578401048937	3/23/2020 12:00:0	☐
	Pocohey	White	Computers	.30904206834219	.50627248757812	4/27/2020 12:00:0	☑
	Pocohey	Green	Washers	.65251446825107	.95011441826361	7/22/2020 12:00:0	☐
	Pocohey	Green	Stoves	.57267136106861	395840506486515	5/9/2021 12:00:00	☐
	Macko	Green	Computers	.38359617925425	769301403206448	11/29/2020 12:00	☐
	Pocohey	Blue	Stoves	.05875172934435	.30913512981923	7/8/2020 12:00:00	☐
	Studeby	Red	Cars	10.299093006318	.33809615128584	4/24/2020 12:00:0	☐
	Macko	Red	Washers	427893281182222	336410643922356	3/17/2020 12:00:0	☐
	Pocohey	White	Cars	0.2998116800095	53.563661824243	10/26/2020 12:00	☐
	Macko	Red	Stoves	.78990361457215	388760731736554	12/10/2020 12:00	☐
	Pocohey	Red	Stoves	.46993096148128	.39809247450813	2/5/2021 12:00:00	☑
	Pocohey	White	Washers	.19248118400225	.37984173576342	3/5/2021 12:00:00	☐

アプリケーションの設定

1. 新しい Windows Forms アプリケーションを作成し、プロジェクトのプロパティウィンドウを使用してプロジェクトフレームワークを .NET 6.0 に設定します。
2. **NuGetパッケージマネージャー**を使用して**C1.Win.FlexGrid.ja**パッケージをインストールします。パッケージがインストールされると、C1FlexGridコントロールがツールボックスに追加されます。
3. FlexGridコントロールをツールボックスからWindowsフォームアプリケーションにドラッグアンドドロップするか、FlexGridコントロールを初期化し、次のコードを使用してフォームに追加します。

C#

```
// コントロールを初期化します。
C1FlexGrid flexGrid = new C1FlexGrid();
// フォームにコントロールを追加します。
this.Controls.Add(flexGrid);
```

VB.NET

```
' コントロールを初期化します。
Dim flexGrid As C1FlexGrid = New C1FlexGrid()
' フォームにコントロールを追加します。
Me.Controls.Add(flexGrid)
```

Note: WinForms Edition supports WinForms designer in Visual Studio 2019 version 16.11.0 and higher.

データソースへのFlexGridの連結

1. データソースとして使用するカスタムクラス「Product」を作成します。

C#

```
// カスタムクラスProductを作成します。
public class Product
{
    static Random _rnd = new Random();
```

```
    static string[] _names = "Macko|Surfair|Pocohey|Studeby".Split('|');
    static string[] _lines = "Computers|Washers|Stoves|Cars".Split('|');
    static string[] _colors = "Red|Green|Blue|White".Split('|');
    public Product()
    {
        Name = _names[_rnd.Next() % _names.Length];
        Line = _lines[_rnd.Next() % _lines.Length];
        Color = _colors[_rnd.Next() % _colors.Length];
        Price = 30 + _rnd.NextDouble() * 1000;
        Cost = 3 + _rnd.NextDouble() * 300;
        Discontinued = _rnd.NextDouble() < .2;
        Introduced = DateTime.Today.AddDays(_rnd.Next(-600, 0));
    }
    public string Name { get; set; }
    public string Color { get; set; }
    public string Line { get; set; }
    public double Price { get; set; }
    public double Cost { get; set; }
    public DateTime Introduced { get; set; }
    public bool Discontinued { get; set; }
}
```

VB.NET

```
' カスタムクラスProductを作成します。
Public Class Product
    Private Shared _rnd As Random = New Random()
    Private Shared _names As String() =
"Macko|Surfair|Pocohey|Studeby".Split("|"c)
    Private Shared _lines As String() =
"Computers|Washers|Stoves|Cars".Split("|"c)
    Private Shared _colors As String() = "Red|Green|Blue|White".Split("|"c)

    Public Sub New()
        Name = _names(_rnd.[Next]() Mod _names.Length)
        Line = _lines(_rnd.[Next]() Mod _lines.Length)
        Color = _colors(_rnd.[Next]() Mod _colors.Length)
        Price = 30 + _rnd.NextDouble() * 1000
        Cost = 3 + _rnd.NextDouble() * 300
        Discontinued = _rnd.NextDouble() < .2
        Introduced = Date.Today.AddDays(_rnd.[Next](-600, 0))
    End Sub

    Public Property Name As String
    Public Property Color As String
    Public Property Line As String
    Public Property Price As Double
    Public Property Cost As Double
    Public Property Introduced As Date
    Public Property Discontinued As Boolean
End Class
```

2. Product 型のリストを初期化します。

C#

```
// Productがクラス型であるProduct型のリストを初期化します。
```

```
List<Product> _products = new List<Product>();
```

VB.NET

```
' Productがクラス型であるProduct型のリストを初期化します。
Dim _products As List(Of Product) = New List(Of Product)()
```

3. For ループを初期化し、製品をリストに追加します。

C#

```
// forループを初期化し、製品をリストに追加します。
for (int i = 0; i <100; i++)
{
    _products.Add(new Product());
}
```

VB.NET

```
' forループを初期化し、製品をリストに追加します。
For i As Integer = 0 To 100 - 1
    _products.Add(New Product())
Next
```

4. 作成されたデータソースに FlexGrid を連結します。

C#

```
// FlexGridにデータを連結します
flexGrid.DataSource = _products;
```

VB.NET

```
' FlexGridにデータを連結します
flexGrid.DataSource = _products
```

FlexGrid コントロールの構成

以下のコードを追加して、実行時にグリッドとその列を構成します。

C#

```
c1FlexGrid1.Styles.Fixed.ForeColor = Color.Blue;
c1FlexGrid1.Styles.Fixed.Font = new Font("Microsoft Sans serif", 9, FontStyle.Bold);
c1FlexGrid1.Dock = DockStyle.Fill;
c1FlexGrid1.Cols[6].Format = "c";
```

VB.NET

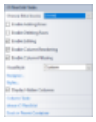
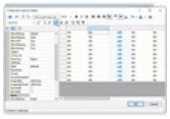
```
c1FlexGrid1.Styles.Fixed.ForeColor = Color.Blue
c1FlexGrid1.Styles.Fixed.Font = New Font("Microsoft Sans serif", 9, FontStyle.Bold)
c1FlexGrid1.Dock = DockStyle.Fill
c1FlexGrid1.Cols(6).Format = "c"
```

Note: WinForms .NET 6 Edition does not include rich design-time support yet. We will enhance it in future releases.

設計時サポート

FlexGrid for WinForms は、プログラミング作業を容易にするためにさまざまな設計時オプションを備えています。スマートタグ、コンテキストメニュー、プロパティグリッドに加えて、FlexGrid は、タスクメニューとエディタも提供しています。設計時に、グリッド全体に対しては **C1FlexGrid タスク**メニューを使用し、グリッドの各列でよく使用するプロパティに対しては**列タスク**メニューに設定オプションがあります。同時に、**C1FlexGrid 列エディタ**、**C1FlexGrid スタイルエディタ**などのさまざまなエディタから、列とスタイルをカスタマイズするための詳細なプロパティが提供されます。

このセクションでは、FlexGrid コントロールに用意されているさまざまな設計時オプションについて説明します。

トピック	スナップショット	コンテンツ
タスクメニュー		C1FlexGrid タスクメニュー、列タスクメニュー、およびそれらのオプションについて説明します。 • C1FlexGrid タスクメニュー • C1FlexGrid 列タスクメニュー
エディタ		さまざまなエディタとそのアクセス方法について説明します。 • C1FlexGrid 列エディタ • C1FlexGrid スタイルエディタ • キャプションスタイルエディタ • 列スタイルエディタ

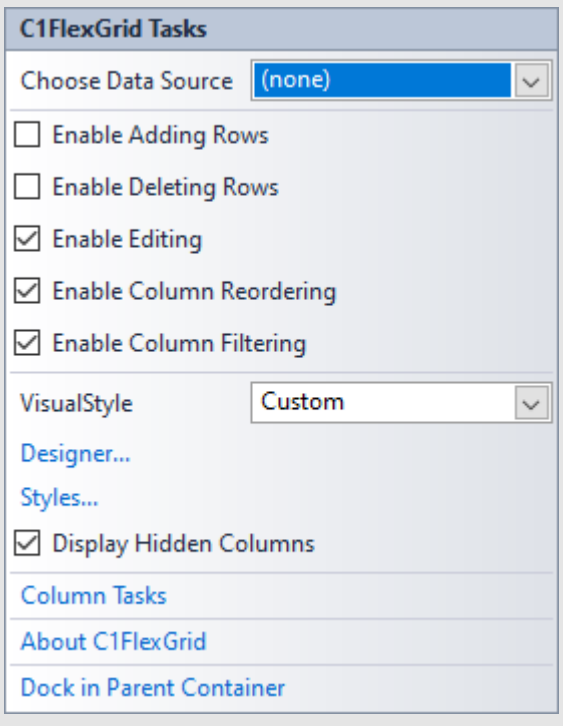
Note: WinForms .NET 6 Edition does not include rich design-time support yet. We will enhance it in future releases.

タスクメニュー

C1FlexGrid タスクメニュー

C1FlexGrid タスクメニューから、よく使用される FlexGrid のプロパティに簡単にアクセスできます。また、**C1FlexGrid 列エディタ**、**C1FlexGrid スタイルエディタ**などのエディタを開くオプションが提供されています。

C1FlexGrid タスクメニューにアクセスするには、グリッドの右上隅にあるスマートタグ(▶)をクリックします。以下の表で、C1FlexGrid タスクメニューにあるさまざまなオプションについて説明します。



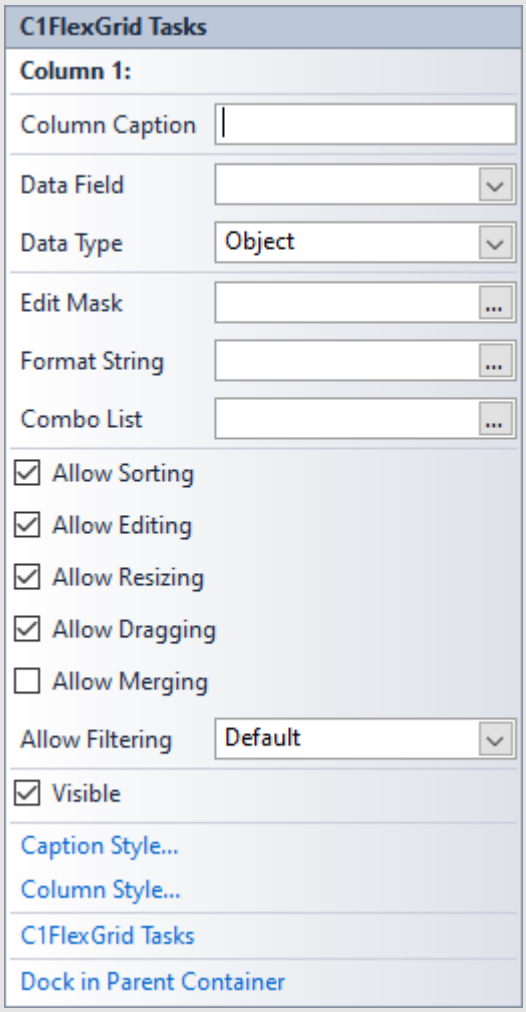
オプション	説明
データソースの選択	ドロップダウンにより、使用可能なデータのリストが開きます。**[プロジェクトデータソースの追加]** オプションで **データソース構成ウィザードを開くことができます。**. 新しいデータソースをプロジェクトに追加する方法については、「連結モード」を参照してください。
行の追加可能	このチェックボックスにより、実行時にグリッドに新しい行を追加できるようにする **AllowAddNew** プロパティが切り替わります。 オンにすると、グリッドの下端に星付きのテンプレート行が表示され、新しいレコードを入力できます。
行の削除可能	このチェックボックスにより、[Del]キーを押して選択した行を削除できるようにする **AllowDelete** プロパティが切り替わります。
編集	このチェックボックスにより、グリッドを編集可能にする **AllowEditing** プロパティが切り替わります。

	可能
列の並べ替え可能	このチェックボックスにより、**AllowDragging** プロパティに **Columns** を設定して、列ヘッダーをドラッグすることによって列の順序を変更可能にします。
列フィルタ処理可能	このチェックボックスにより、グリッド列のフィルタ処理を有効または無効にする **FlexGrid.AllowFiltering** プロパティが切り替わります。
VisualStyle	ドロップダウンで、グリッドで使用可能な組み込みのビジュアルスタイルを選択できます。デフォルトでは、この値は **Custom** に設定されています。
デザイナ	このオプションは、グリッドの各列のプロパティを設定するための C1FlexGrid 列エディタを開きます。
スタイル	このオプションは、設計時にさまざまな定義済みスタイルをカスタマイズしたり、新しいスタイルを作成するための C1FlexGrid スタイルエディタを開きます。
非表示列を表示	このチェックボックスは、デザインビューでグリッド列を表示または非表示にします。このオプションは、実行時の列の表示/非表示には関係しないことに注意してください。そのため、このチェックボックスがオンになっている場合でも、**Visible** プロパティに **False** が設定されている列は、アプリケーションの実行時に表示されません。
列タスク	このオプションは、タスクメニューを **C1FlexGrid 列タスク**メニューに切り替えます。これで、選択した列のプロパティを設定するオプションが提供されます。
C1FlexGrid について	このオプションは、FlexGrid コントロールのバージョンなどの情報を表示するダイアログボックスを表示します。
親コンテナにドッキングする	このオプションは、グリッドの Dock プロパティに **Fill** を設定します。これで、フォームスペース全体を占有するようにグリッドのサイズが変更されます。また、このオプションをクリックするとテキストが切り替わり、**[親コンテナでドッキングを解除する]**オプションになります。これは、グリッドを元のサイズに戻します。

C1FlexGrid 列タスクメニュー

C1FlexGrid 列タスクメニューから、よく使用されるグリッド列のプロパティに簡単にアクセスできます。また、**[キャプションスタイル]**および**[列スタイル]**という名前のエディタを開くオプションが提供されます。

C1FlexGrid 列タスクメニューにアクセスするには、設計時に構成された列のヘッダーをダブルクリックします。C1FlexGrid 列タスクメニューを開くもう 1 つの方法として、グリッドの右上隅のスマートタグ (▶) をクリックし、さらに**[列タスク]**オプションに移動します。以下の表で、C1FlexGrid 列タスクメニューにあるさまざまなオプションについて説明します。



オプション	説明
列キャプション	このフィールドでは、列のヘッダーセルにテキストを設定する **Caption** プロパティの値を指定できます。
データフィールド	このドロップダウンで、データソース内のフィールドを選択するためのリストを開き、列の **Name** プロパティの値を設定します。
データ型	ドロップダウンで、使用可能なデータ型のリストを開き、選択した列のデータ型を設定します。このフィールドは、列の **DataType** プロパティに対応しています。
編集マスク	このフィールドでは、選択した列のマスクを設定する **EditMask** プロパティの値を指定します。フィールドの右側の省略符ボタンを押すと**[入力マスク]**ダイアログボックスが開き、定義済みマスクのリストからマスクを選択できます。
書式文字列	このフィールドでは、ソースからデータ値を表示するための書式文字列を設定する **Format** プロパティの値を指定します。
コンボリスト	このフィールドでは、ユーザーが選択できる複数の値のリストを指定します。フィールドの右側の省略符ボタンを押すと**[コンボリスト]**ダイアログボックスが開き、そこで値オプションを指定できます。

ソートを許可	このチェックボックスは、**AllowSorting** プロパティを切り替えて、列のソートを有効または無効にします。デフォルトでは、すべての列に対してソートが有効です。
編集を許可	このチェックボックスは、**AllowEditing** プロパティを切り替えて、列を読み取り専用または編集可能にします。デフォルトでは、すべての列に対して編集が有効です。
サイズ変更を許可	このチェックボックスは、**AllowResizing** プロパティを切り替えて、実行時に列のサイズを調整できるようにします。デフォルトでは、すべての列に対してサイズ変更が有効です。
ドラッグを許可	このチェックボックスは、**AllowDragging** プロパティを切り替えて、実行時に列の順序を変更できるようにします。デフォルトでは、すべての列に対してドラッグが有効です。
結合を許可	このチェックボックスは、**AllowMerging** プロパティを切り替えて、同じ値を持ち、隣接する 2 つの列のセルの結合を有効または無効にします。デフォルトでは、すべての列に対して結合は無効です。
フィルタを許可	このドロップダウンで、**Column.AllowFiltering** プロパティの値を指定して、各列のフィルタのタイプを選択します。使用できるオプションには、**Default**、**ByValue**、**ByCondition**、**Custom**、**None** があります。
表示	このチェックボックスで、列の **Visible** プロパティの値を切り替えて、実行時にグリッド内の列の表示/非表示を設定します。

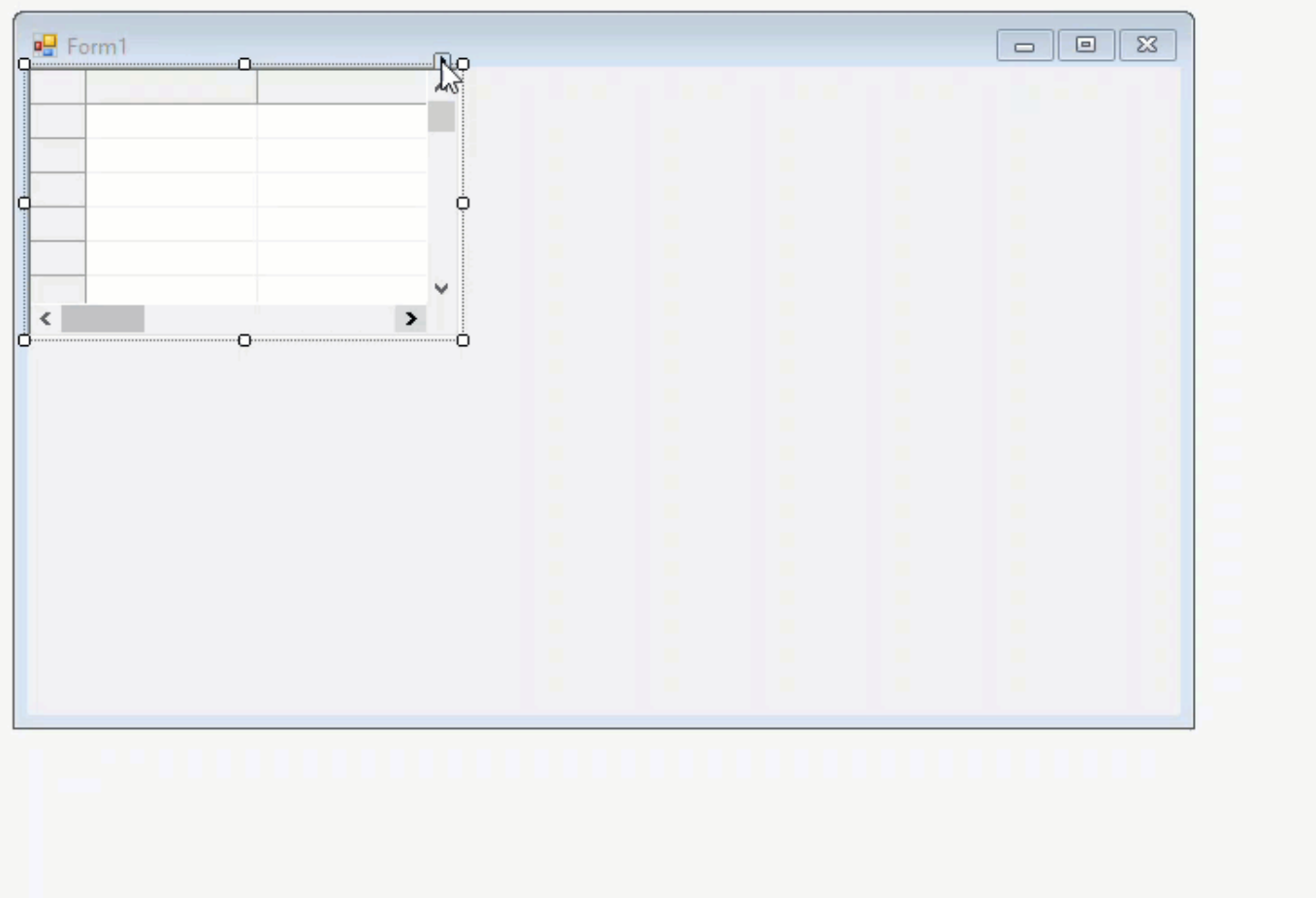
キャプションスタイル	このオプションは、キャプションスタイルエディタを開き、列ヘッダーセルのテキスト、配置、背景、および境界線のスタイルを設定するオプションを提供します。
列スタイル	このオプションは、列スタイルエディタを開き、列テキストのテキスト、配置、背景、および境界線のスタイルを設定するオプションを提供します。
C1FlexGrid のタスク	このオプションは、タスクメニューを **C1FlexGrid タスク**メニューに切り替えます。これで、グリッド全体のプロパティを設定するオプションが提供されます。
親コンテナにドッキングする	このオプションは、グリッドの Dock プロパティに **Fill** を設定します。これで、フォーム全体を占有するようにグリッドのサイズが変更されます。また、このオプションをクリックするとテキストが切り替わり、**[親コンテナでドッキングを解除する]**オプションになります。これは、**Dock** プロパティに **None** を設定して、グリッドを元のサイズに戻します。

エディタ

タスクメニューとは別に、FlexGrid には、グリッド、その列、および列のスタイルに関連する数多くのプロパティを設定するためのさまざまなエディタが用意されています。

C1FlexGrid 列エディタ

FlexGrid の **C1FlexGrid 列エディタ**を使用すると、コードを記述することなく、設計時に簡単に列の作業を行うことができます。このエディタの左側にあるプロパティペインからは、**Caption**、**DataType**、**AllowFiltering**、**AllowMerging** など、列に関連するさまざまなプロパティを設定できます。このエディタの上部のツールバーには、データソースからのデータの再ロード、列の追加、削除、サイズ変更などのオプションがあります。このツールバーのオプションを使用して、列テキストのスタイルを設定することもできます。列テキストをさらにカスタマイズするには、**列スタイル**エディタを使用して、詳細なプロパティにアクセスできます。

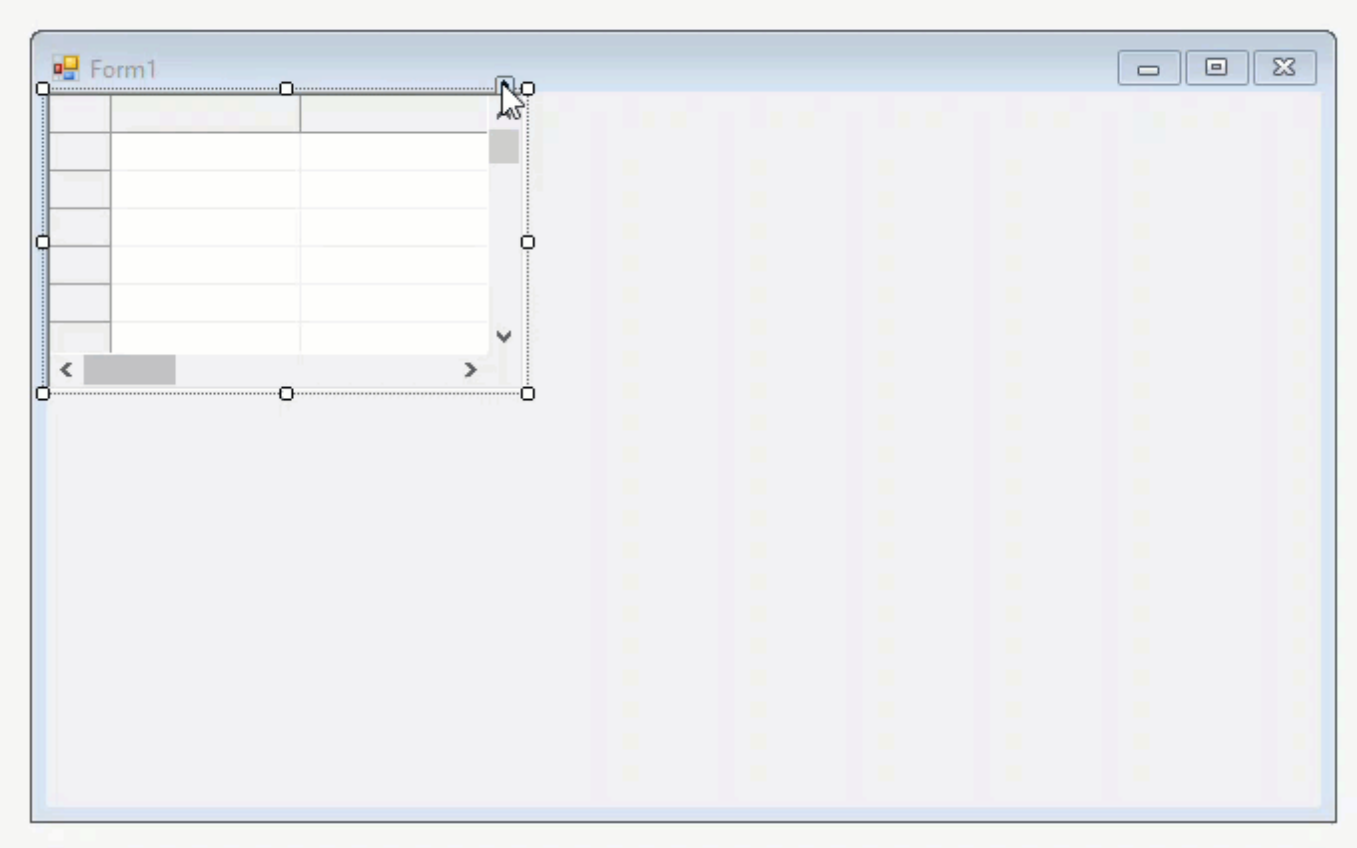


設計時に、C1FlexGrid 列エディタには次の 3 つの方法でアクセスできます。

- **スマートタグ**:グリッドの右上隅のスマートタグ(▶)をクリックし、**C1FlexGrid タスク**メニューから**[デザイナ]**を選択します。
- **プロパティウィンドウ**:グリッドを選択し、**プロパティ**ウィンドウに移動して、**Cols** プロパティの横にある省略符ボタン(**...**)をクリックします。
- **コンテキストメニュー**:グリッドを右クリックし、コンテキストメニューから**[デザイナ]**を選択します。

C1FlexGrid スタイルエディタ

FlexGrid の **C1FlexGrid スタイル**エディタを使用して、組み込みスタイルをカスタマイズしたり、新しいスタイルを作成することで、設計時にグリッドのスタイルを設定できます。このエディタの左側には、特定のタイプのセルに対するスタイルがリストされ、右側のプロパティウィンドウで、それらのスタイルをカスタマイズできます。スタイルのリストは、デフォルトの**[組み込みスタイル]**と**[カスタムスタイル]**の 2 つのセクションに分かれています。その下には、カスタムスタイルの追加と削除を行う**[追加]**ボタンと**[削除]**ボタン、およびエディタをデフォルト設定に戻す**[クリア]**ボタンがあります。下端の**[自動書式設定]**ボタンを押すと、2 つ目の**[C1FlexGrid 自動書式設定]**ダイアログが開きます。ここには、定義済みのスタイルのセットが表示され、**[プレビュー]**ペインにプレビューが表示されます。



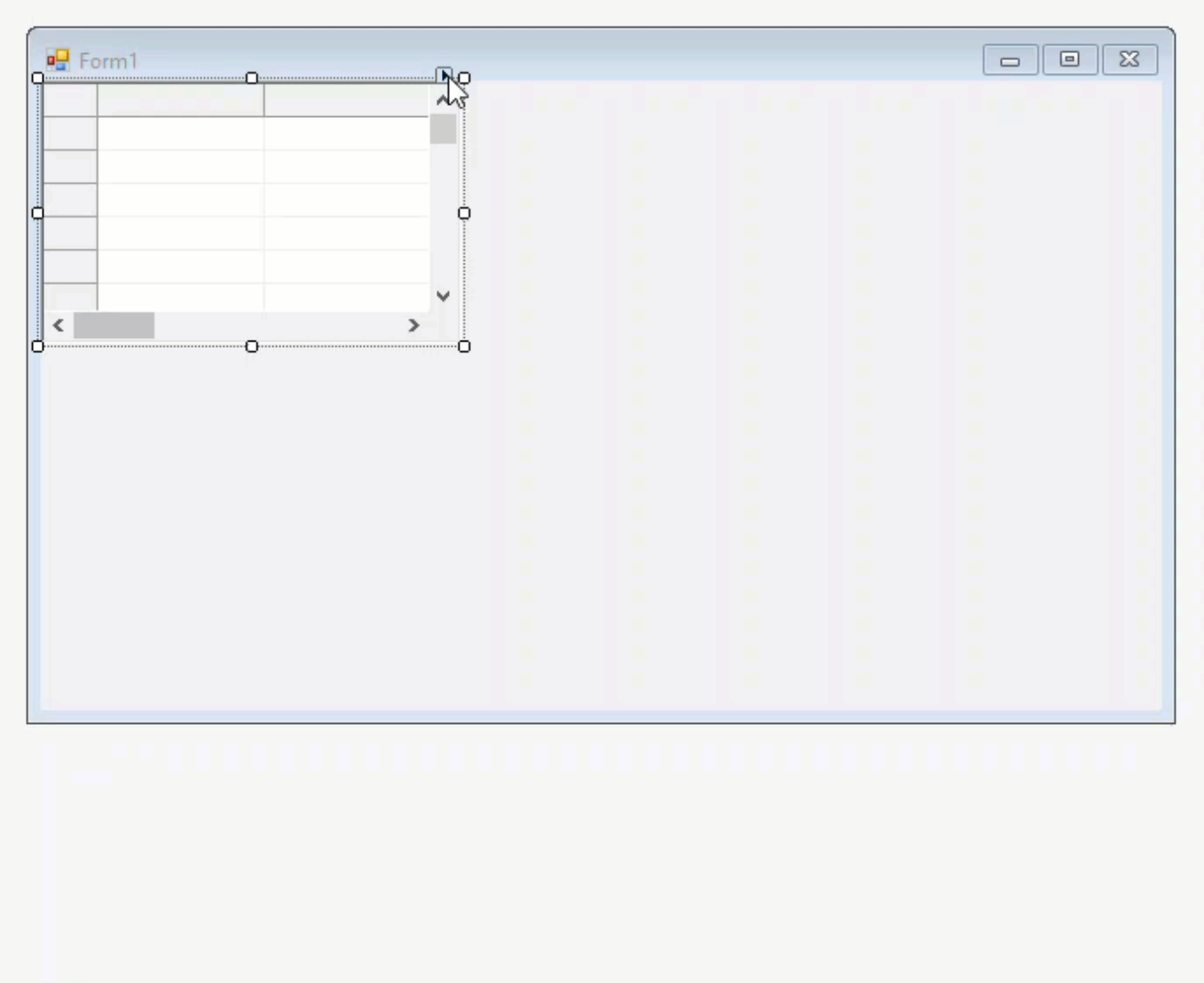
設計時に、C1FlexGrid スタイルエディタには次の 3 つの方法でアクセスできます。

- **スマートタグ**:グリッドの右上隅のスマートタグ(▶)をクリックし、**C1FlexGrid タスク**メニューから**[スタイル]**を選択します。
- **プロパティウィンドウ**:グリッドを選択し、**プロパティ**ウィンドウに移動して、**Styles** プロパティの横にある省略符ボタン(**...**)をクリックします。
- **コンテキストメニュー**:グリッドを右クリックし、コンテキストメニューから**[スタイル]**を選択します。

キャプションスタイルエディタ

FlexGrid の**キャプションスタイル**エディタを使用すると、列ヘッダーセルとそのテキストのスタイル関連プロパティを設定できます。このエディタには、ヘッダーセルのさまざまな部分をカスタマイズするための 4 つのタブ、**[テキスト]**、**[配置]**、**[境界線]**、および**[背景]**があります。

キャプションスタイルエディタにアクセスするには、スマートタグ(▶)をクリックし、タスクメニューの**[C1FlexGrid 列タスク]**オプションをクリックして列タスクを開き、次に**[キャプションスタイル]**オプションに移動します。



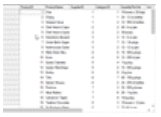
列スタイルエディタ

キャプションスタイルエディタと同様に、FlexGrid には、列テキストのスタイルを設定するための**列スタイル**エディタもあります。このエディタには、**キャプションスタイル**エディタと同じオプションがあります。唯一の違いは、キャプションスタイルエディタは列ヘッダーテキストをカスタマイズするのに対して、このエディタは列セル内の一般的なテキストに使用されます。

列スタイルエディタにアクセスするには、スマートタグ（▶）をクリックし、タスクメニューの**[C1FlexGrid 列タスク]**オプションをクリックして列タスクを開き、次に**[列スタイル]**オプションに移動します。

データ

このセクションでは、FlexGrid コントロールにデータを挿入するさまざまな方法について説明します。

トピック	スナップショット	コンテンツ
非連結モード		非連結グリッドにデータを挿入する方法について説明します。
連結モード		データの連結によってグリッドにデータを挿入する方法と、連結されたグリッドに対するさまざまな操作について説明します。 • 設計時の連結 • DataSource プロパティを使用した実行時の連結 • SetDataBinding メソッドを使用した実行時の連結 • 連結モードの操作

非連結モード

名前が示すように、非連結モードでは、グリッドに連結されるデータソースはなく、データはコントロール自体に格納されます。この場合、データを提供するには、設計時に行と列を追加するか、行と列のコレクションを通してプログラムで追加する必要があります。また、空のグリッドを作成し、ユーザーにデータを入力してもらうこともできます。

グリッドがデータを格納せず、ユーザーが手動でデータを管理する必要があることから、非連結グリッドはあまり一般的な方法ではありません。ただし、ビジネスシナリオによっては、レコードの作成や保守などに非連結グリッドが適することもあります。たとえば、非連結モードでグリッドを使用して、毎日の販売データを記録したり、日単位の在庫の変化を管理することができます。次の例は、コードを通してデータが挿入されたグリッドを具体的に示しています。

	ID	Category	Item	Revenue	City	Region
	1140	Mobile	Iphone XR	2334091	Macau	South China
	1140	Mobile	Iphone XR	109300	Haikou	South China
	1140	Mobile	Iphone XR	1734621	Jinan	North China
	1894	Mobile	OnePlus 7Pro	499100	Beijing	North China
	1894	Mobile	OnePlus 7Pro	459000	Haikou	South China
	1252	Mobile	Samsung S9	896250	Beijing	North China
	1252	Mobile	Samsung S9	435064	Haikou	South China
	1252	Mobile	Samsung S9	716520	Macau	South China
	3489	Appliances	Haier 394L 4Star	367050	Beijing	North China
	3489	Appliances	Haier 394L 4Star	578900	Sanya	South China

C1FlexGrid では、行または列オブジェクトの Count プロパティを設定することで、空の行または列を追加できます。また、これらのコレクションの Add メソッドを使用して、グリッドに空の行と列を追加できます。セルにデータを設定するには、使い慣れたインデックス表記（Item プロパティ）か、SetData メソッドを使用できます。セルへのデータの設定の詳細については、「データの格納と取得」を参照してください。

以下のコードを使用して、非連結の FlexGrid for WinForms にデータを挿入します。

C#

```
// 連結されていない列を追加します
Column col = _flex.Cols.Add();
col.Name = col.Caption = "Unbound";
_flex[1, "Unbound"] = 123;
```

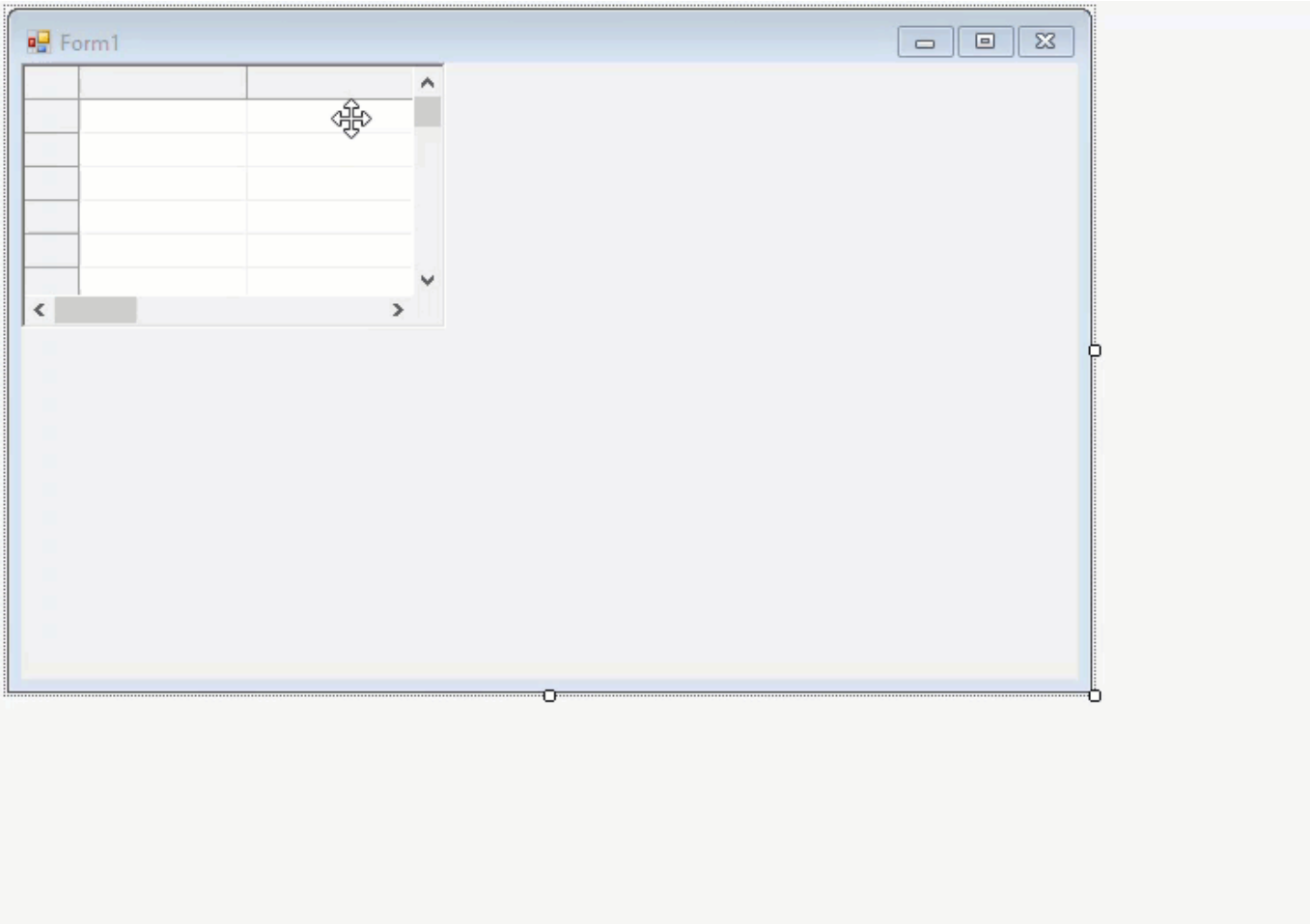
VB.NET

```
' 連結されていない列を追加します
Dim col As Column = C1FlexGrid1.Cols.Add()
col.Name = "Unbound"
col.Caption = "Unbound"
C1FlexGrid1(1, "Unbound") = 123
```

連結モード

連結モードは、その名前が示すように、グリッドが基盤のデータソースからデータを取得する状態を指します。また、データ連結では、複数のデータコンシューマーが同期的にデータプロバイダと接続できます。

FlexGrid は、ObservableCollection、IList<T>、List<T>、Array、BindingSource、ADO.NET オブジェクト（DataSet、DataTable など）のような一般に使用される大部分のデータソースに対するデータ連結をサポートします。



FlexGrid とデータソースを連結するには、次の 3 つの方法があります。

- 設計時の連結
- DataSource プロパティを使用した実行時の連結
- SetDataBinding メソッドを使用した実行時の連結

設計時の連結

1. デザインビューで、FlexGrid コントロールを選択し、スマートタグをクリックして**C1FlexGrid タスク**メニューを開きます。
2. **[データソースの選択]**ドロップダウンボタンをクリックし、**[プロジェクトデータソースの追加]**オプションを選択して**データソース構成ウィザード**を開きます。
3. **[データソースの種類の選択]** ページで、**[データベース]**を選択し、**[次へ]**をクリックします。
4. **[データベースモデルの選択]**ページで、**[データセット]**を選択し、**[次へ]**をクリックします。
5. **[データ接続の選択]**ページで、**[新しい接続]** をクリックして **[接続の追加]** ダイアログを開きます。
6. **[データソース]**フィールドに対して、**[選択]**ボタンをクリックして **[データソースの変更]**ダイアログを開きます。
7. **[Microsoft Access データベースファイル]**を選択し、**[OK]** をクリックして[接続の追加]ダイアログに戻ります。
8. **[データベースファイル名]**フィールドに対して、**[参照]**をクリックして、データベースファイルに移動します。データベースファイルへの接続に必要な場合は、**ユーザー名とパスワード**を指定します。この例では、デフォルトで次の場所にある **C1NWind.mdb** ファイルを使用します。

\Documents\ComponentOne Samples\Common

9. **[接続のテスト]**をクリックしてデータベースまたはサーバーに正しく接続されていることを確認し、**[OK]**をクリックします。
10. **[OK]**をクリックして[接続の追加]ダイアログボックスを閉じます。
11. **[次へ]**をクリックして続行します。データファイルをプロジェクトに追加し、接続文字列を修正するかどうかを確認するダイアログボックスが表示されます。要件に従って適切なオプションを選択します。
12. **[次の名前で接続を保存する]**ボックスをオンにして名前を入力して、接続文字列をアプリケーション構成ファイルに保存します。
13. **[次へ]**をクリックして**[データベースオブジェクトの選択]**ページに切り替えます。
14. **[テーブル]**ノードから、たとえば **Products** テーブルを選択し、**[終了]**をクリックします。
15. 以上の手順で、プロジェクトにデータセットと接続文字列が追加されます。また、Visual Studio は、データセットを設定するための以下のコードを自動的に作成します。

C#

```
// 次のサンプルコードは、データを「c1NWindDataSet.Products」テーブルにロードします。 必要に応じて、移動または削除できます。
this.productsTableAdapter.Fill(this.c1NWindDataSet.Products);
```

VB.NET

```
' 次のサンプルコードは、データを「c1NWindDataSet.Products」テーブルにロードします。 必要に応じて、移動または削除できます。
Me.ProductsTableAdapter.Fill(Me.c1NWindDataSet.Products)
```

DataSource プロパティを使用した実行時の連結

FlexGrid は、実行時に FlexGrid コントロールとデータソースを連結するために DataSource プロパティを提供しています。FlexGrid の **DataSource** プロパティには、データソースオブジェクトを割り当てる必要があります。データソースオブジェクトに 2 つ以上のテーブルがある場合は、DataMember プロパティにテーブル名を設定して、目的のテーブルを指定できます。

以下のコードは、DataSource プロパティを使用してデータを WinForms FlexGrid に連結します。

C#

```
c1FlexGrid1.BeginUpdate();
C1DataCollection<Customer> c1DataCollection = new C1DataCollection<Customer>
(GetData());
c1FlexGrid1.DataSource = new C1DataCollectionBindingList(c1DataCollection);
c1FlexGrid1.EndUpdate();
```

VB.NET

```
c1FlexGrid1.BeginUpdate()
Dim c1CollectionView As C1.DataCollection.C1DataCollection(Of Customer) = New
C1.DataCollection.C1DataCollection(Of Customer)(GetData())
c1FlexGrid1.DataSource = New
C1.DataCollection.BindingList.C1DataCollectionBindingList(c1CollectionView)
c1FlexGrid1.EndUpdate()
```

上記のサンプルコードは、**GetData** という名前のカスタムメソッドを使用してデータを提供しています。要件に基づいてデータソースを設定できます。以下のコードは、WinForms FlexGrid のデータを作成する例を示しています。

C#

```
ObservableCollection<Customer> GetData()
{
    var data = new ObservableCollection<Customer>();
```

```
    for (int i = 0; i < 25; i++)
        data.Add(new Customer());
    return data;
}
```

VB.NET

```
Private Function GetData() As ObservableCollection(Of Customer)
    Dim data = New ObservableCollection(Of Customer)()

    For i = 0 To 25 - 1
        data.Add(New Customer())
    Next

    Return data
End Function
```

SetDataBinding メソッドを使用した実行時の連結

FlexGrid には SetDataBinding メソッドもあり、これを使用して **DataSource** プロパティと **DataMember** プロパティの両方を 1 回の呼び出しで設定できます。以下のコードは、**SetDataBinding** メソッドを使用して、親テーブルといくつかの子テーブルを別々のグリッドにレンダリングする例を具体的に示しています。

以下のコードは、SetDataBinding メソッドを使用して、データを WinForms FlexGrid に連結する方法を具体的に示しています。

C#

```
// グリッドにデータセットを連結します
_flexCustomers.BeginUpdate();
_flexCustomers.SetDataBinding(ds, "Customers");
_flexCustomers.EndUpdate();

_flexOrders.BeginUpdate();
_flexOrders.SetDataBinding(ds, "Customers.CustomerOrders");
_flexOrders.EndUpdate();

_flexOrderDetails.BeginUpdate();
_flexOrderDetails.SetDataBinding(ds, "Customers.CustomerOrders.OrderDetails");
_flexOrderDetails.EndUpdate();
```

VB.NET

```
' グリッドにデータセットを連結します
_flexCustomers.BeginUpdate()
_flexCustomers.SetDataBinding(ds, "Customers")
_flexCustomers.EndUpdate()

_flexOrders.BeginUpdate()
_flexOrders.SetDataBinding(ds, "Customers.CustomerOrders")
_flexOrders.EndUpdate()

_flexOrderDetails.BeginUpdate()
_flexOrderDetails.SetDataBinding(ds, "Customers.CustomerOrders.OrderDetails")
_flexOrderDetails.EndUpdate()
```

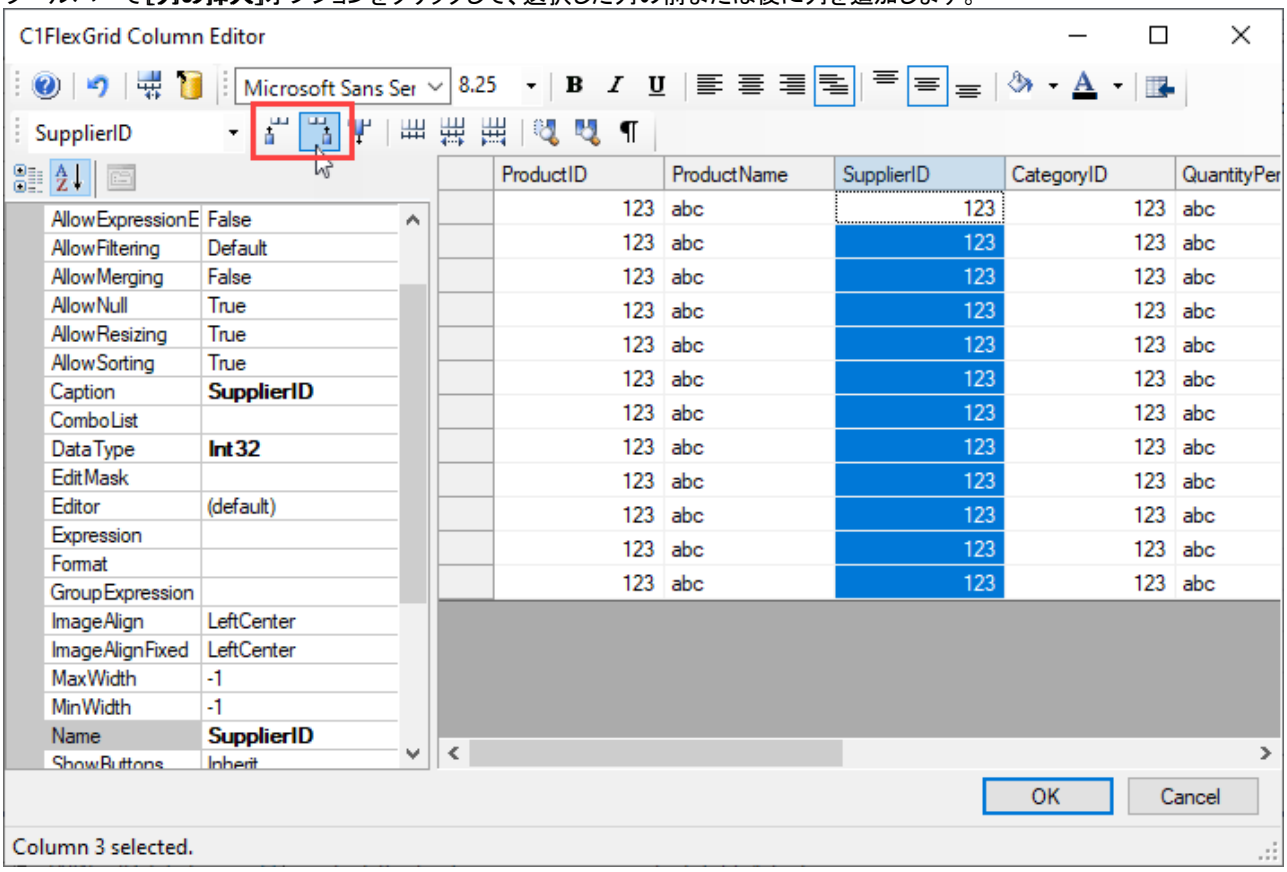
連結モードの操作

連結モードでの非連結列の追加

連結モードのグリッドは、データソースからデータを取得し、それをレコードおよび連結列として表示します。列を追加してデータソースに存在しないデータを表示するには、設計時またはコードから非連結列を追加する必要があります。

設計時の非連結列の追加

1. デザインビューで、FlexGrid コントロールを選択し、スマートタグをクリックして**C1FlexGrid タスク**メニューを開きます。
2. グリッドコントロールとデータソースを連結します。FlexGrid とデータソースを連結する手順については、「連結モード」を参照してください。
3. **[デザイナ]**オプションをクリックして、**C1FlexGrid 列エディタ**を開きます。
4. 右側のペインで、グリッドプレビューから既存の列を選択します。
5. ツールバーで**[列の挿入]**オプションをクリックして、選択した列の前または後に列を追加します。



コードからの非連結列の追加

FlexGrid は、グリッドに非連結列を追加するために、ColumnCollection クラスの Add、Insert、およびInsertRange メソッドを提供しています。**Add** メソッドは末尾に列を追加します。**Insert** メソッドを使用すると、新しい列を追加する位置を指定できます。同様に、**InsertRange** メソッドを使用して、指定した位置に複数の列を追加できます。

以下のコードを使用して、連結 WinForms FlexGrid に非連結列を追加します。

C#

```
// 連結されていない列を追加します
Column col = _flex.Cols.Add();
col.Name = col.Caption = "Unbound";
_flex[1, "Unbound"] = 123;
```

VB.NET

```
' 連結されていない列を追加します
Dim col As Column = C1FlexGrid1.Cols.Add()
col.Name = "Unbound"
col.Caption = "Unbound"
```

```
C1FlexGrid1(1, "Unbound") = 123
```

非連結列の値の設定

連結グリッドの非連結列に値を定義するには、C1FlexGrid クラスの SetUnboundValue イベントと GetUnboundValue イベントを使用する必要があります。最初に、入力値を格納するためのハッシュテーブルを作成します。次に、**SetUnboundValue** イベントを使用し、レコードを識別する一意キーを使用してハッシュテーブルに非連結値を設定します。さらに、**GetUnboundValue** イベントを使用し、ハッシュテーブルに格納された値を一意キーで取得して、セルに表示する非連結値を設定します。

以下のコードは、連結 WinForms FlexGrid に追加された非連結列に値を設定する方法を示します。

C#

```
// get value from hashtable using ProductID as key
void _flex_GetUnboundValue(object sender, C1.Win.C1FlexGrid.UnboundValueEventArgs e)
{
       DataRowView drv = (DataRowView)_flex.Rows[e.Row].DataSource;
   e.Value = _hash[drv["ProductID"]];
}

// store value in hashtable using ProductID as key
void _flex_SetUnboundValue(object sender, C1.Win.C1FlexGrid.UnboundValueEventArgs e)
{
       DataRowView drv = (DataRowView)_flex.Rows[e.Row].DataSource;
        _hash[drv["ProductID"]] = e.Value;
}
```

VB.NET

```
Private Sub C1FlexGrid1_GetUnboundValue(sender As Object, e As
C1.Win.C1FlexGrid.UnboundValueEventArgs) Handles C1FlexGrid1.GetUnboundValue
    Dim drv As DataRowView = DirectCast(C1FlexGrid1.Rows(e.Row).DataSource, DataRowView)
    e.Value = _hash(drv("ProductID"))
End Sub

Private Sub C1FlexGrid1_SetUnboundValue(sender As Object, e As
C1.Win.C1FlexGrid.UnboundValueEventArgs) Handles C1FlexGrid1.SetUnboundValue
    Dim drv As DataRowView = DirectCast(C1FlexGrid1.Rows(e.Row).DataSource, DataRowView)
    _hash(drv("ProductID")) = e.Value
End Sub
```

固定列のデータの表示

固定列に値を設定するには、フォームロードイベントで C1FlexGrid クラスの **DrawMode** プロパティに **OwnerDraw** を設定し、次に **OwnerDrawCell** イベントを作成して固定列セルに値を設定します。この例では、連結 WinForms FlexGrid の固定列に行番号を設定しています。

	ProductName	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock
1	Chai	1	10 boxes x 20 bags	18	39
2	Chang	1	24 - 12 oz bottles	19	17
3	Aniseed Syrup	2	12 - 550 ml bottles	10	13
4	Chef Anton's Cajun	2	48 - 6 oz jars	22	53
5	Chef Anton's Gumb	2	36 boxes	21.35	0
6	Grandma's Boysenl	2	12 - 8 oz jars	25	120
7	Uncle Bob's Organ	7	12 - 1 lb pkgs.	30	15
8	Northwoods Cranb	2	12 - 12 oz jars	40	6
9	Mishi Kobe Niku	6	18 - 500 g pkgs.	97	29
10	Ikura	8	12 - 200 ml jars	31	31
11	Queso Cabrales	4	1 kg pkg.	21	22

C#

```
    private void C1FlexGrid1_OwnerDrawCell(object sender, OwnerDrawCellEventArgs e)

    {
      if ((e.Row >= this.c1FlexGrid1.Rows.Fixed) & (e.Col == (this.c1FlexGrid1.Cols.Fixed
- 1)))
       {
         e.Text = e.Row.ToString(); // または任意のテキスト
       }
    }
```

VB.NET

```
Private Sub C1FlexGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e As
OwnerDrawCellEventArgs)
    If e.Row >= Me.c1FlexGrid1.Rows.Fixed And e.Col = Me.c1FlexGrid1.Cols.Fixed - 1 Then
        e.Text = e.Row.ToString() ' または任意のテキスト
    End If
End Sub
```

列幅の自動調整

FlexGrid には、テキスト長に合わせて自動的に列幅を調整するために、**C1FlexGrid** クラスの **AutoResize** プロパティがあります。データソースに連結して適切な列幅でグリッドをロードするには、このプロパティに **true** を設定しておく必要があります。**AutoSizeCol** メソッドを呼び出して、指定した列の幅を調整することもできます。

以下のコードを使用して、WinForms FlexGrid でテキストに合わせて列幅を自動調整します。

C#

```
    // すべての列の幅を自動的に調整します
    c1FlexGrid1.AutoResize = true;

    // 4列目の幅を自動調整します
    // c1FlexGrid1.AutoSizeColumns(3);
```

VB.NET

```
  ' すべての列の幅を自動的に調整します
  c1FlexGrid1.AutoResize = True

  ' 4列目の幅を自動調整します
  ' c1FlexGrid1.AutoSizeColumns(3)
```

フィールドへのビットマップ画像の表示

大部分のシナリオでは、グリッドは、データソースから画像を直接取得します。ただし、Microsoft Access のように画像を OLE オブジェクトとして格納するデータベースにグリッドが連結されている場合は、ビットマップ画像を取得するために多少余分な処理が必要です。このような場合は、バイト配列として格納された画像データをメモリストリームに変換し、次に **OwnerDrawCell** イベントを使用して画像を読み込む必要があります。フォームロードイベントで、**DrawMode** プロパティに **OwnerDraw** を設定する必要があります。また、画像を適切に表示するために行の高さを調整します。

以下のコードは、WinForms FlexGrid のフィールドにビットマップ画像を表示する方法を具体的に示しています。

C#

```
    private void C1FlexGrid1_OwnerDrawCell(object sender,
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
     {
       if(!e.Measuring && e.Row>=c1FlexGrid1.Rows.Fixed && e.Col >=
```

```
c1FlexGrid1.Cols.Fixed)
        {
         // 「Photo」はblob(byte [])に保存された画像です
         if (c1FlexGrid1.Cols[e.Col].Name == "Picture")
              {
                // mdbからロードしてみてください
                e.Image = LoadImage(c1FlexGrid1[e.Row, e.Col] as byte[]);

               // 画像が表示された場合は、テキストを入力しないでください
                if (e.Image != null) e.Text = null;
                }
            }
        }
```

VB.NET

```
Private Sub C1FlexGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs)
    If Not e.Measuring AndAlso e.Row >= c1FlexGrid1.Rows.Fixed AndAlso e.Col >=
c1FlexGrid1.Cols.Fixed Then

        ' 「Photo」はblob(byte [])に保存された画像です
        If c1FlexGrid1.Cols(e.Col).Name Is "Picture" Then
            ' mdbからロードしてみてください
            e.Image = LoadImage(TryCast(c1FlexGrid1(e.Row, e.Col), Byte()))

            ' 画像が表示された場合は、テキストを入力しないでください
            If e.Image IsNot Nothing Then e.Text = Nothing
        End If
    End If
End Sub
```

上のサンプルコードでは、カスタムメソッドの LoadImage メソッドを使用して、画像をバイト配列からメモリストリームに変換しています。

C#

```
Image LoadImage(byte[] picData)
   static Image LoadImage(byte[] picData)
   {
      // これが埋め込みオブジェクトであることを確認してください
        const int bmData = 78;
        if (picData == null || picData.Length < bmData + 2) return null;
        if (picData[0] != 0x15 || picData[1] != 0x1c) return null;

      // 現在点ではビットマップのみを処理します
         if (picData[bmData] != 'B' || picData[bmData + 1] != 'M') return null;

     // 画像をロードします
        Image img = null;
          try
            {
                MemoryStream ms = new MemoryStream(picData, bmData, picData.Length -
bmData);
                img = Image.FromStream(ms);
            }
            catch { }

      // 画像を返します
         return img;
        }
```

VB.NET

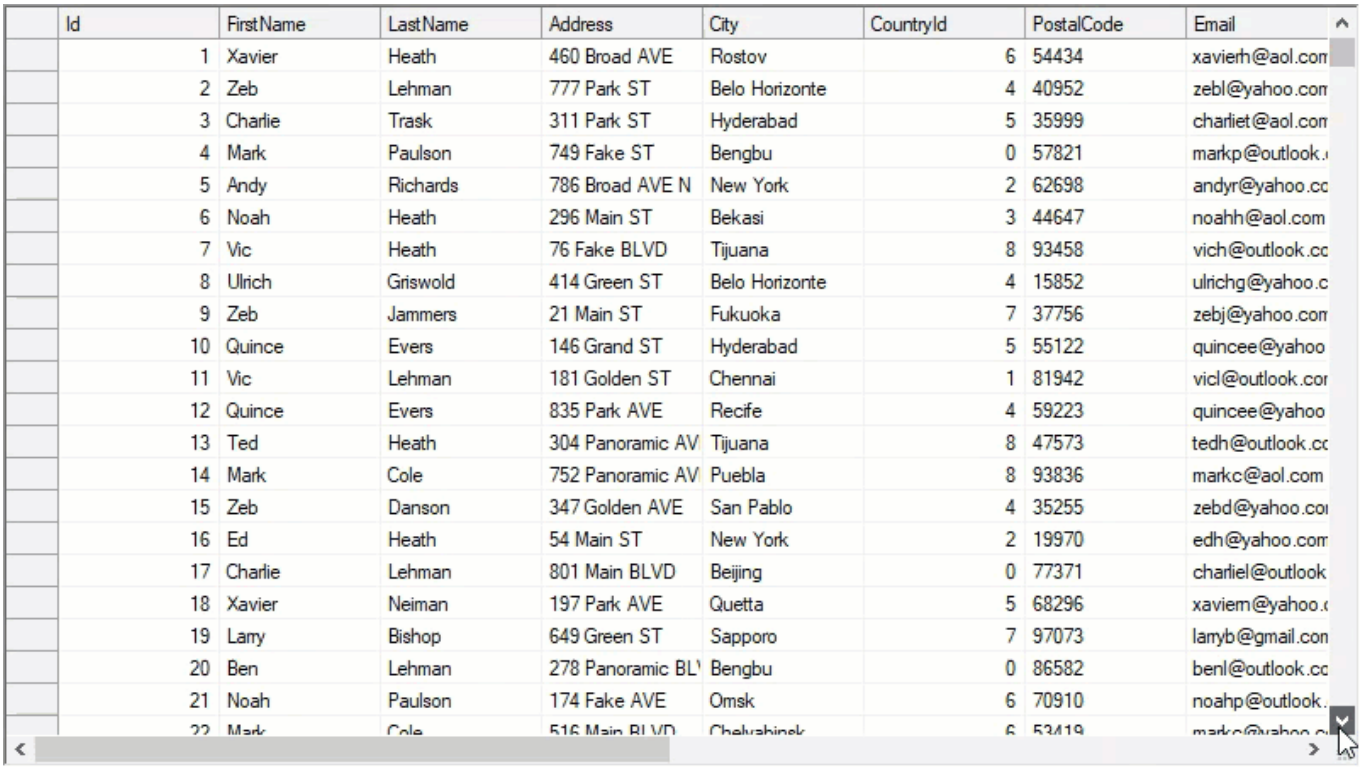
```
Private Function LoadImageMethod(ByVal picData As Byte()) As Image
```

```
Private Shared Function LoadImage(ByVal picData As Byte()) As Image
  ' これが埋め込みオブジェクトであることを確認してください
  Const bmData As Integer = 78
  If picData Is Nothing OrElse picData.Length < bmData + 2 Then Return Nothing
  If picData(0) <> &H15 OrElse picData(1) <> &H1c Then Return Nothing

  ' 現在点ではビットマップのみを処理します
  If picData(bmData) <> "B"c OrElse picData(bmData + 1) <> "M"c Then Return Nothing

  ' 画像をロードします
  Dim img As Image = Nothing

  Try
    Dim ms As MemoryStream = New MemoryStream(picData, bmData, picData.Length - bmData)
    img = Image.FromStream(ms)
  Catch
  End Try

  ' 画像を返します
  Return img
End Function
```

データの仮想化

Data virtualization refers to the process of loading data from remote sources in incremental manner, instead of loading it all at once. That is, in virtual mode, data is loaded on demand in chunks as the user scrolls the grid. Hence, the process is really useful while dealing with huge volumes of data and enables efficient loading of data in a shorter period of time.

There are two widely adopted approaches for data virtualization, pagination and cursor. In Pagination approach, a limit and offset parameter are send to the remote source to specify which portion of the data is requested. This approach returns the total amount of items available, so that the client may know how to retrieve the rest of the items. While, in Cursor approach, a token is send initially, which is null at first, to fetch the pages sequentially. In this case, the received page contains the token to the next one.



Id	FirstName	LastName	Address	City	CountryId	PostalCode	Email
1	Xavier	Heath	460 Broad AVE	Rostov	6	54434	xavierh@aol.com
2	Zeb	Lehman	777 Park ST	Belo Horizonte	4	40952	zebl@yahoo.com
3	Charlie	Trask	311 Park ST	Hyderabad	5	35999	charliet@aol.com
4	Mark	Paulson	749 Fake ST	Bengbu	0	57821	markp@outlook.c
5	Andy	Richards	786 Broad AVE N	New York	2	62698	andyr@yahoo.co
6	Noah	Heath	296 Main ST	Bekasi	3	44647	noahh@aol.com
7	Vic	Heath	76 Fake BLVD	Tijuana	8	93458	vich@outlook.co
8	Ulrich	Griswold	414 Green ST	Belo Horizonte	4	15852	ulrichg@yahoo.c
9	Zeb	Jammers	21 Main ST	Fukuoka	7	37756	zebj@yahoo.com
10	Quince	Evers	146 Grand ST	Hyderabad	5	55122	quincee@yahoo
11	Vic	Lehman	181 Golden ST	Chennai	1	81942	vicl@outlook.com
12	Quince	Evers	835 Park AVE	Recife	4	59223	quincee@yahoo
13	Ted	Heath	304 Panoramic AV	Tijuana	8	47573	tedh@outlook.co
14	Mark	Cole	752 Panoramic AV	Puebla	8	93836	markc@aol.com
15	Zeb	Danson	347 Golden AVE	San Pablo	4	35255	zebd@yahoo.com
16	Ed	Heath	54 Main ST	New York	2	19970	edh@yahoo.com
17	Charlie	Lehman	801 Main BLVD	Beijing	0	77371	charliel@outlook
18	Xavier	Neiman	197 Park AVE	Quetta	5	68296	xaviern@yahoo.c
19	Larry	Bishop	649 Green ST	Sapporo	7	97073	larryb@gmail.com
20	Ben	Lehman	278 Panoramic BL	Bengbu	0	86582	benl@outlook.co
21	Noah	Paulson	174 Fake AVE	Omsk	6	70910	noahp@outlook.
22	Mark	Cole	516 Main BLVD	Chelyabinsk	6	53419	markc@yahoo.co

FlexGrid supports data virtualization and gives excellent performance while loading large data sets in real time. To

implement virtual mode in the FlexGrid control, we use C1DataCollection library which provides data virtualization for any datagrid or list control.

The C1DataCollection namespace provides C1VirtualDataCollection and C1CursorDataCollection to implement pagination and cursor approach respectively. Both classes have an abstract **GetPageAsync()** method that must be implemented to return the items that will populate the collection. The base classes take the responsibility of caching the data as well as dispatching the requests of the pages according to a series of parameters that prevents too many pages to be requested together. In this topic, we are explaining the data virtualization and its implementation using the **C1VirtualDataCollection** class.

Implement Data Virtualization

First of all, implement the **C1VirtualDataCollection** interface to override the GetPageAsync method as per your data. The GetPageAsync method of C1VirtualDataCollection returns the items in the page as well as a token to the next page. This method uses pageIndex, startingIndex, count, sortDescriptions, filterExpression and cancellation Token as parameters. The parameters pageIndex denotes the index of the requesting page, startingIndex denotes the index where the returned items will be inserted and count denotes the number of items to be returned. The **GetPageAsync** method returns total number of items (TotalCount) along with a list of the requested number of items (count) starting from an index (startingIndex).

C#

```
public class VirtualModeCollectionView : C1VirtualDataCollection<Customer>
{
    public int TotalCount { get; set; } = 1_000;

    protected override async Task<Tuple<int, IReadOnlyList<Customer>>>
GetPageAsync(int pageIndex, int startingIndex, int count,
IReadOnlyList<SortDescription> sortDescriptions = null, FilterExpression
filterExpression = null, CancellationToken cancellationToken =
default(CancellationToken))
    {
        await Task.Delay(500, cancellationToken);//Simulates network traffic.
        return new Tuple<int, IReadOnlyList<Customer>>(TotalCount,
Enumerable.Range(startingIndex, count).Select(i => new Customer(i)).ToList());
    }
}
```

VB.NET

```
Public Class VirtualModeCollectionView
    Inherits C1VirtualDataCollection(Of Customer)

    Public Property TotalCount As Integer = 1_000

    Protected Overrides Async Function GetPageAsync(ByVal pageIndex As Integer,
ByVal startingIndex As Integer, ByVal count As Integer, ByVal Optional
sortDescriptions As IReadOnlyList(Of SortDescription) = Nothing, ByVal Optional
filterExpression As FilterExpression = Nothing, ByVal Optional cancellationToken As
CancellationToken = Nothing) As Task(Of Tuple(Of Integer, IReadOnlyList(Of
Customer)))
        Await Task.Delay(500, cancellationToken)
        Return New Tuple(Of Integer, IReadOnlyList(Of Customer))(TotalCount,
Enumerable.Range(startingIndex, count).[Select](Function(i) New
Customer(i)).ToList())
    End Function


End Class
```

Apply Data Virtualization to FlexGrid

To apply data virtualization on FlexGrid and populate it with the virtual data collection, create an object of C1DataCollectionBindingList passing an instance of **VirtualModeCollectionView** and assign it to **DataSource** property of the grid.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    var collectionView = new VirtualModeCollectionView();
    c1FlexGrid1.DataSource = new C1DataCollectionBindingList(collectionView);
    c1FlexGrid1.Visible = true;
}
```

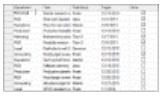
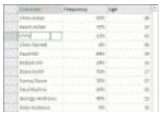
VB.NET

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim collectionView As New VirtualModeCollectionView
    Dim data = New C1DataCollectionBindingList(collectionView)
    C1FlexGrid1.DataSource = New C1DataCollectionBindingList(collectionView)
    C1FlexGrid1.Visible = True
End Sub
```

列

グリッドコントロールは、行と列のコレクションです。一般に、列にはそれぞれ特定の種類の情報が格納されます。一方、行は、それぞれ特定の項目に関するさまざまな種類の情報を記録するために使用されます。

FlexGrid では、列のコレクションは ColumnCollection クラスで表されます。これにアクセスするには、C1FlexGrid クラスの Cols プロパティを使用します。このセクションでは、列で実行できるさまざまな操作について説明します。

トピック	スナップショット	コンテンツ
基本的な操作		基本的な列の操作方法について説明します。 • 列数の設定 • 列の追加 • 列の削除 • 列の挿入 • データ型の設定 • 固定列の設定 • フリーズ列の設定
ユーザー操作		実行時にエンドユーザーが実行できる操作について説明します。 • 編集の許可 • ドラッグの許可 • フリーズの許可 • ソートの許可 • フィルタの許可 • サイズ変更の許可
エディタ		FlexGrid のさまざまな組み込みエディタと、エディタに関連する操作について説明します。また、FlexGrid でのカスタムエディタの使用方法についても説明します。 • チェックボックス • 数値 • 日付 • コンボボックス • マスク • マップリスト • セルボタン
検証		FlexGrid のセルに検証を適用する方法、およびエラー情報を表示する方法について説明します。
スパークライン		FlexGrid のセルにスパークラインチャートを表示する方法について説明します。
ヘッダーとフッター		列ヘッダーと列フッターの設定方法について説明します。 • ヘッダーテキストの設定 • 列ヘッダーの結合 • 列ヘッダーテキストの折り返し • 列ヘッダーのスタイル設定 • 列フッターの設定
サイズ変更		列のサイズを変更するさまざまな方法について説明します。 • 列幅の設定 • 列幅の自動調整 • 最小/最大列幅の設定 • スターサイズ

基本的な操作

このトピックでは、列で実行可能なさまざまな基本操作について説明します。

列数の設定

グリッドがデータソースに連結されている場合、列の数はデータソースにあるフィールドの数によって決まります。ただし、非連結モードの場合は、ColumnCollection クラスの Count プロパティに任意の値を設定して、グリッドに表示する列の数を指定できます。

非連結モードで WinForms FlexGrid に追加する列の数を設定するには、次のコードを使用します。

C#

```
// 列数を設定します
c1FlexGrid1.Cols.Count = 4;
```

VB.NET

```
' 列数を設定します
c1FlexGrid1.Cols.Count = 4
```

列の追加

FlexGrid で **ColumnCollection** クラスを使用して新しい列を追加する方法は 2 つあります。新しい列を追加するには、Add メソッドを使用するか、このクラスにある **Count** プロパティの値を増やします。これらのメソッドを使用して連結グリッドに新しい列を追加した場合、それは単なる非連結列として追加されます。

次に、列を WinForms FlexGrid に追加する 2 つの方法を示します。

C#

```
// 方法 1:
// 列コレクションのAddメソッドを使用します
C1.Win.C1FlexGrid.Column c;
c = c1FlexGrid1.Cols.Add();
c.Caption = "New Column";

// 方法 2:
// 列コレクションのCountプロパティを変更します
c1FlexGrid1.Cols.Count += 1;
c1FlexGrid1.Cols[c1FlexGrid1.Cols.Count - 1].Caption = "New Column";
```

VB.NET

```
' 方法 1:
' 列コレクションのAddメソッドを使用します
Dim c As C1.Win.C1FlexGrid.Column
c = c1FlexGrid1.Cols.Add()
c.Caption = "New Column"

' 方法 2:
' 列コレクションのCountプロパティを変更します
c1FlexGrid1.Cols.Count += 1
c1FlexGrid1.Cols(c1FlexGrid1.Cols.Count - 1).Caption = "New Column"
```

列の削除

グリッドから特定の列を削除するには、**ColumnCollection** クラスの Remove メソッドを使用して、削除する列をそのパラメータとして指定します。非連結グリッドでは、**Count** プロパティの値を変更することで、列の数を減らすこともできます。こうすると、列コレクションの最後から列が削除されます。**ColumnCollection** クラスには、列の範囲を削除できる **RemoveRange** メソッドもあります。

次のコードスニペットは、WinForms FlexGrid から列を削除する 3 つの方法をすべて示しています。

C#

```
// 方法 1:
// 列コレクションのRemoveメソッドを使用して2番目の列を削除します
c1FlexGrid1.Cols.Remove(2);

// 方法 2:
// Countプロパティを変更して、最後の列を削除します
c1FlexGrid1.Cols.Count -= 1;

// 方法 3:
// RemoveRangeメソッドを使用して、2番目から始まる4つの列を削除します
c1FlexGrid1.Cols.RemoveRange(2, 4);
```

VB.NET

```
' 方法 1:
' 列コレクションのRemoveメソッドを使用して2番目の列を削除します
c1FlexGrid1.Cols.Remove(2)

' 方法 2:
' Countプロパティを変更して、最後の列を削除します
c1FlexGrid1.Cols.Count -= 1

' 方法 3:
' RemoveRangeメソッドを使用して、2番目から始まる4つの列を削除します
c1FlexGrid1.Cols.RemoveRange(2, 4)
```

列の挿入

FlexGrid の特定の位置に列を挿入するには、**ColumnCollection** クラスの **Insert** メソッドを使用し、このメソッドで列を挿入する位置を指定します。また、グリッドに複数の列を挿入するには、**InsertRange** メソッドを使用します。グリッドが連結グリッドの場合でも、これらのメソッドでは非連結列のみが追加されます。

列を WinForms FlexGrid の特定の位置に挿入するには、次のいずれかの方法を使用します。

C#

```
C1.Win.C1FlexGrid.Column c;

// 方法 1:
// Insertメソッドを使用して2番目の位置に列を挿入します
c = c1FlexGrid1.Cols.Insert(2);
c.Caption = "Inserted Column";

// 方法 2:
// InsertRangeメソッドを使用して2番目の位置に3つの列を挿入します
// c1FlexGrid1.Cols.InsertRange(2, 3);
```

VB.NET

```
Dim c As C1.Win.C1FlexGrid.Column

' 方法 1:
' Insertメソッドを使用して2番目の位置に列を挿入します
c = c1FlexGrid1.Cols.Insert(2)
c.Caption = "Inserted Column"
```

```
' 方法 2:
' InsertRangeメソッドを使用して2番目の位置に3つの列を挿入します
' c1FlexGrid1.Cols.InsertRange(2, 3)
```

データ型の設定

連結 FlexGrid の場合、各連結列のデータ型は、データに応じてデータソースから自動的に取得されます。ただし、非連結モードの場合は、**Column** クラスの **DataType** プロパティを指定して、列のデータ型を設定できます。また、**列タスク**メニューを使用して、設計時に **DataType** プロパティを設定することもできます。タスクメニューの詳細については、「タスクメニュー」を参照してください。FlexGrid の列に特定のデータ型が設定されている場合、セルは、そのデータ型が受け入れ可能なキー入力のみを受け付けます。たとえば、数値型のセルにアルファベットを入力することはできません。

次のコードに示すように、WinForms FlexGrid の非連結列のデータ型を設定します。

C#

```
// 最初の列のデータ型を設定します
c1FlexGrid1.Cols[1].DataType = typeof(int);
```

VB.NET

```
' 最初の列のデータ型を設定します
c1FlexGrid1.Cols(1).DataType = GetType(Integer)
```

固定列の設定

固定列は、編集不可のセルを含む列です。ユーザーがグリッドを水平方向にスクロールしても、固定列はグリッドの左側に常に表示されます。FlexGrid で固定列を設定するには、**ColumnCollection** クラスの **Fixed** プロパティを使用します。このプロパティは、固定する列の数を指定する整数値を受け取ります。

WinForms FlexGrid で 1 つの列を固定列として設定するには、次のコードを使用します。

C#

```
// 1つの列を固定列として設定します
c1FlexGrid1.Cols.Fixed = 1;
```

VB.NET

```
' 1つの列を固定列として設定します
c1FlexGrid1.Cols.Fixed = 1
```

フリーズ列の設定

フリーズ列は、固定列と同様にスクロールできません。ただし、ユーザーはこの列を編集できます。FlexGrid でフリーズ列を設定するには、**ColumnCollection** クラスにある **Frozen** プロパティを使用します。

WinForms FlexGrid でフリーズ列を設定するには、次のコードを使用します。

C#

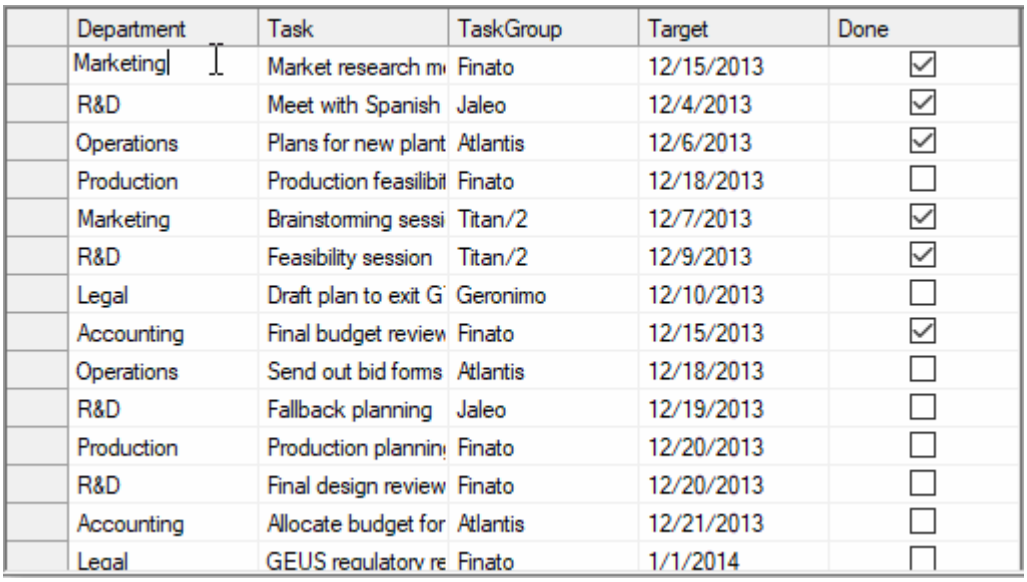
```
// 最初の4列を静止として設定します
c1FlexGrid1.Cols.Frozen = 4;
```

VB.NET

```
' 最初の4列を静止として設定します
c1FlexGrid1.Cols.Frozen = 4
```

ユーザー操作

このトピックでは、エンドユーザーに FlexGrid の列の操作を許可する方法について説明します。



	Department	Task	TaskGroup	Target	Done
	Marketing	Market research m	Finato	12/15/2013	☑
	R&D	Meet with Spanish	Jaleo	12/4/2013	☑
	Operations	Plans for new plant	Atlantis	12/6/2013	☑
	Production	Production feasibilit	Finato	12/18/2013	☐
	Marketing	Brainstorming sessi	Titan/2	12/7/2013	☑
	R&D	Feasibility session	Titan/2	12/9/2013	☑
	Legal	Draft plan to exit G	Geronimo	12/10/2013	☐
	Accounting	Final budget review	Finato	12/15/2013	☑
	Operations	Send out bid forms	Atlantis	12/18/2013	☐
	R&D	Fallback planning	Jaleo	12/19/2013	☐
	Production	Production plannin	Finato	12/20/2013	☐
	R&D	Final design review	Finato	12/20/2013	☐
	Accounting	Allocate budget for	Atlantis	12/21/2013	☐
	Legal	GEUS regulatory re	Finato	1/1/2014	☐

編集の許可

FlexGrid では、デフォルトで、グリッドのすべての列を編集できます。ただし、Column オブジェクトの AllowEditing プロパティを **false** に設定することで、特定の列の編集を無効にすることができます。また、**C1FlexGrid.AllowEditing** プロパティを **false** に設定すると、グリッド全体の編集を無効にできます。編集の詳細については、「編集」を参照してください。

次のコードは、WinForms FlexGrid の編集を無効にする一方で、1 つの列は編集可能のまま残す方法を示しています。

C#

```
// グリッド全体で編集を無効にします
c1FlexGrid1.AllowEditing = false;

// 3列目でのみ編集を有効にします
c1FlexGrid1.Cols[3].AllowEditing = true;
```

VB.NET

```
' グリッド全体で編集を無効にします
c1FlexGrid1.AllowEditing = False

' 3列目でのみ編集を有効にします
c1FlexGrid1.Cols(3).AllowEditing = True
```

ドラッグの許可

デフォルトでは、ユーザーが列ヘッダーをドラッグして目的の位置にドロップすることで、列を並べ替えることができます。ただし、FlexGrid.AllowDragging プロパティと Column.AllowDragging プロパティを使用して、この動作を変更できます。特定の列のドラッグを無効にするには、その列の **Column.AllowDragging** プロパティを **false** に設定します。一方、**FlexGrid.AllowDragging** プロパティを **Rows** または **None** に設定すると、列の並べ替えがグリッドレベルで無効になり

ます。このプロパティは、AllowDraggingEnum に含まれる値を受け取ります。また、設計時に列の並べ替えを無効にするには、**C1FlexGrid タスク**メニューの[**列の順序変更を有効にする**]チェックボックスをオフにします。タスクメニューの詳細については、「タスクメニュー」を参照してください。

次のコードは、WinForms FlexGrid で列のドラッグを有効にする方法を示しています。

C#

```
// グリッド全体ですべての列のドラッグを許可します
c1FlexGrid1.AllowDragging = C1.Win.C1FlexGrid.AllowDraggingEnum.Columns;

// 特定の列のドラッグを無効にします
c1FlexGrid1.Cols[3].AllowDragging = false;
```

VB.NET

```
' グリッド全体ですべての列のドラッグを許可します
c1FlexGrid1.AllowDragging = C1.Win.C1FlexGrid.AllowDraggingEnum.Columns

' 特定の列のドラッグを無効にします
c1FlexGrid1.Cols(3).AllowDragging = False
```

フリーズの許可

実行時にエンドユーザーが列をフリーズできるようにするには、C1FlexGrid クラスの AllowFreezing プロパティを使用します。これは、AllowFreezingEnum に含まれる値を受け取ります。このプロパティが **Columns** または **Both** に設定されている場合は、マウスポインタをヘッダー列の端に置くと鍵のアイコンが表示されます。ユーザーは、鍵のアイコンをクリックしてドラッグすることで列をフリーズできます。

次のコードは、WinForms FlexGrid の列のフリーズをユーザーに許可する方法を示しています。

C#

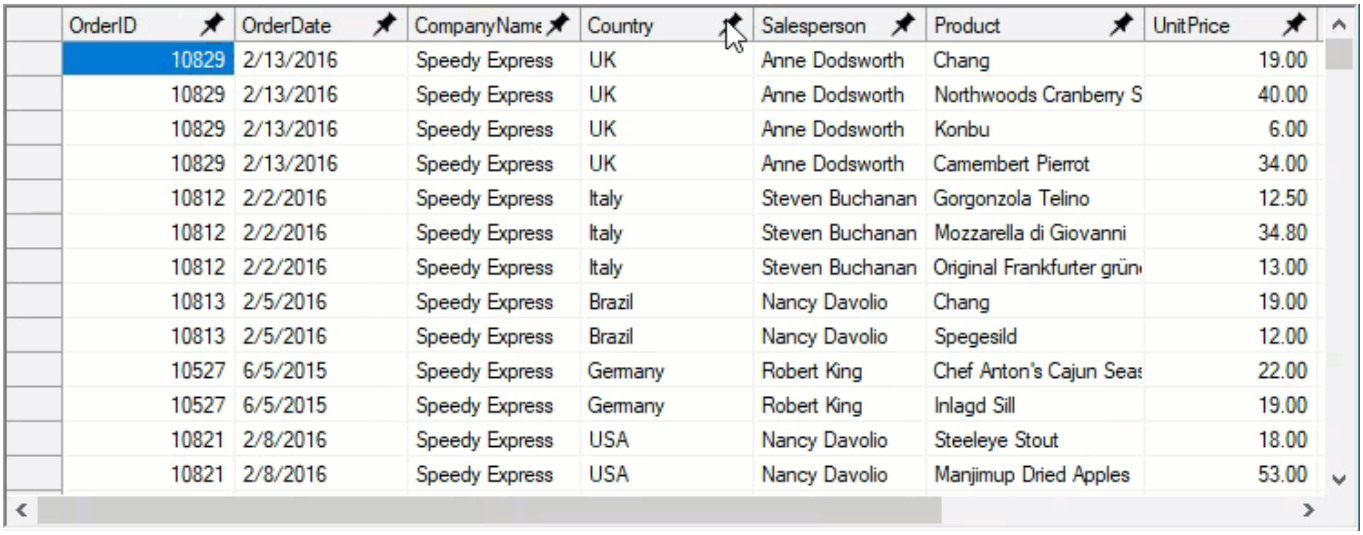
```
// 実行時に列のフリーズを許可します
c1FlexGrid1.AllowFreezing = C1.Win.C1FlexGrid.AllowFreezingEnum.Columns;
```

VB.NET

```
' 実行時に列のフリーズを許可します
c1FlexGrid1.AllowFreezing = C1.Win.C1FlexGrid.AllowFreezingEnum.Columns
```

ピン留めの許可

FlexGrid では、実行時に特定の列または列範囲をピン留めすることをユーザーに許可できます。それには、**C1FlexGrid** クラスの AllowPinning プロパティを使用します。このプロパティを設定すると、列ヘッダーにピンボタン()が追加されます。ユーザーは、このボタンを使用して実行時に列をピン留めし、グリッドを水平方向にスクロールしてもそれらの列がビューに留まるようにすることができます。このプロパティは、AllowPinning 列挙 に含まれる値を受け取ります。これを使用して、1 個の列または列範囲をピン留めできます。このプロパティが **ColumnRange** に設定されている場合、ユーザーは左側の列からクリックした列までのすべての列を 1 回の操作でピン留めまたはピン留め解除できます。



OrderID	OrderDate	CompanyName	Country	Salesperson	Product	UnitPrice
10829	2/13/2016	Speedy Express	UK	Anne Dodsworth	Chang	19.00
10829	2/13/2016	Speedy Express	UK	Anne Dodsworth	Northwoods Cranberry S	40.00
10829	2/13/2016	Speedy Express	UK	Anne Dodsworth	Konbu	6.00
10829	2/13/2016	Speedy Express	UK	Anne Dodsworth	Camembert Pierrot	34.00
10812	2/2/2016	Speedy Express	Italy	Steven Buchanan	Gorgonzola Telino	12.50
10812	2/2/2016	Speedy Express	Italy	Steven Buchanan	Mozzarella di Giovanni	34.80
10812	2/2/2016	Speedy Express	Italy	Steven Buchanan	Original Frankfurter grün	13.00
10813	2/5/2016	Speedy Express	Brazil	Nancy Davolio	Chang	19.00
10813	2/5/2016	Speedy Express	Brazil	Nancy Davolio	Spegesild	12.00
10527	6/5/2015	Speedy Express	Germany	Robert King	Chef Anton's Cajun Seas	22.00
10527	6/5/2015	Speedy Express	Germany	Robert King	Inlagd Sill	19.00
10821	2/8/2016	Speedy Express	USA	Nancy Davolio	Steeleye Stout	18.00
10821	2/8/2016	Speedy Express	USA	Nancy Davolio	Manjimup Dried Apples	53.00

WinForms FlexGrid で、複数の列のピン留めをユーザーに許可するには、次のコードを使用します。

C#

```
// ユーザーが列範囲をピン留めできるようにします。
c1FlexGrid1.AllowPinning = C1.Win.C1FlexGrid.AllowPinning.ColumnRange;
```

VB.NET

```
' ユーザーが列範囲をピン留めできるようにします。
c1FlexGrid1.AllowPinning = C1.Win.C1FlexGrid.AllowPinning.ColumnRange
```

ソートを許可

デフォルトの FlexGrid では、グリッド全体で列のソートが有効になっています。また、**C1FlexGrid** クラスの AllowSorting プロパティの値が **Auto** に設定されています。このモードでは、ユーザーが列ヘッダーをクリックして **1 個の列**を、また[Ctrl]キーを押しながら列ヘッダーをクリックして**複数の列**をソートできます。複数の列をソートすると、グリッドの列ヘッダーのソート方向を示すグリフの横にソートインデックスが表示されます。また、AllowSortingEnum 列挙を使用して、ソートを禁止したり、列のソート方法のみを変更することができます。AllowSortingEnum 列挙では、列の自動ソート、1 個の列のソート、複数の列または列範囲のソートを許可するかどうか、またはソートを禁止するかどうかを選択できます。

WinForms FlexGrid で**複数列のソート**を有効にするには、次のコードを使用します。**AllowSorting** プロパティが **MultiColumn** に設定されている場合、ユーザーは列ヘッダーを続けてクリックするだけで、複数の列をソートできます。

C#

```
// グリッドで複数列の並べ替えを有効にします
c1FlexGrid1.AllowSorting = C1.Win.C1FlexGrid.AllowSortingEnum.MultiColumn;

// 1列のみの並べ替えを無効にします
c1FlexGrid1.Cols[1].AllowSorting = false;
```

VB.NET

```
' グリッドで複数列の並べ替えを有効にします
c1FlexGrid1.AllowSorting = C1.Win.C1FlexGrid.AllowSortingEnum.MultiColumn

' 1列のみの並べ替えを無効にします
c1FlexGrid1.Cols(1).AllowSorting = False
```

特定の列のソートを無効にするには、その列の Column.AllowSorting プロパティを **false** に設定する必要があります。ソートの詳細については、「ソート」を参照してください。

フィルタの許可

デフォルトの FlexGrid では、実行時のフィルタ処理は許可されません。ただし、C1FlexGrid.AllowFiltering プロパティを **true** に設定することで、フィルタ処理を有効にできます。特定の列に適用するフィルタのタイプを定義するには、Column.AllowFiltering プロパティを使用します。これは、AllowFiltering 列挙に含まれる次の値を受け取ります。

AllowFiltering の値	**説明**
Default	**ColumnFilter** タイプのフィルタが自動的に作成されます。このフィルタは、値フィルタと条件フィルタの組み合わせです。
ByValue	**ValueFilter** タイプのフィルタが自動的に作成されます。このフィルタは、選択可能な値のチェックボックスリストを表示します。リストでチェックボックスがオフにされた値は、フィルタによって除外されます。
ByCondition	**ConditionFilter** タイプのフィルタが自動的に作成されます。このフィルタは、条件を構成するために **Equals**、**is Greater than**、**Contains**、**Begins with** などのオプションを提供します。フィルタで **And** 演算子と **Or** 演算子を使用して、2 つの条件を組み合わせることもできます。
Custom	フィルタは自動的に作成されませんが、独自のフィルタを定義し、それを **Column** クラスの **Filter** プロパティに明示的に割り当てることができます。
None	列のフィルタ処理は無効になります。

フィルタ処理の詳細については、「フィルタ」を参照してください。

次のコードは、WinForms FlexGrid でフィルタ処理を有効にする方法とフィルタの適用方法を示しています。

C#

```
// グリッドレベルでのフィルタリングを許可します
c1FlexGrid1.AllowFiltering = true;

// 最初の列に条件フィルタを適用します
c1FlexGrid1.Cols[1].AllowFiltering = C1.Win.C1FlexGrid.AllowFiltering.ByCondition;
```

VB.NET

```
' グリッドレベルでのフィルタリングを許可します
c1FlexGrid1.AllowFiltering = True

' 最初の列に条件フィルタを適用します
c1FlexGrid1.Cols(1).AllowFiltering = C1.Win.C1FlexGrid.AllowFiltering.ByCondition
```

サイズ変更の許可

FlexGrid では、デフォルトで、グリッドのすべての列のサイズを変更できます。この動作を変更するには、**C1FlexGrid** クラスの AllowResizing プロパティを使用します。このプロパティは、AllowResizingEnum 列挙に含まれる値を受け取ります。これを使用して、列、行、またはその両方のサイズを変更したり、どの要素もサイズ変更しないことを指定できます。この列挙では、行、列、またはその両方のサイズを一様に変更することもできます。つまり、1 つの列または行のサイズを変更すると、残りの列と行のサイズも自動的に変更されます。また、FlexGrid にはブール型の Column.AllowResizing プロパティもあります。これを使用して、特定の行または列のサイズ変更を有効/無効にできます。

次のコードは、WinForms FlexGrid の列のサイズ変更をユーザーに許可する方法を示しています。

C#

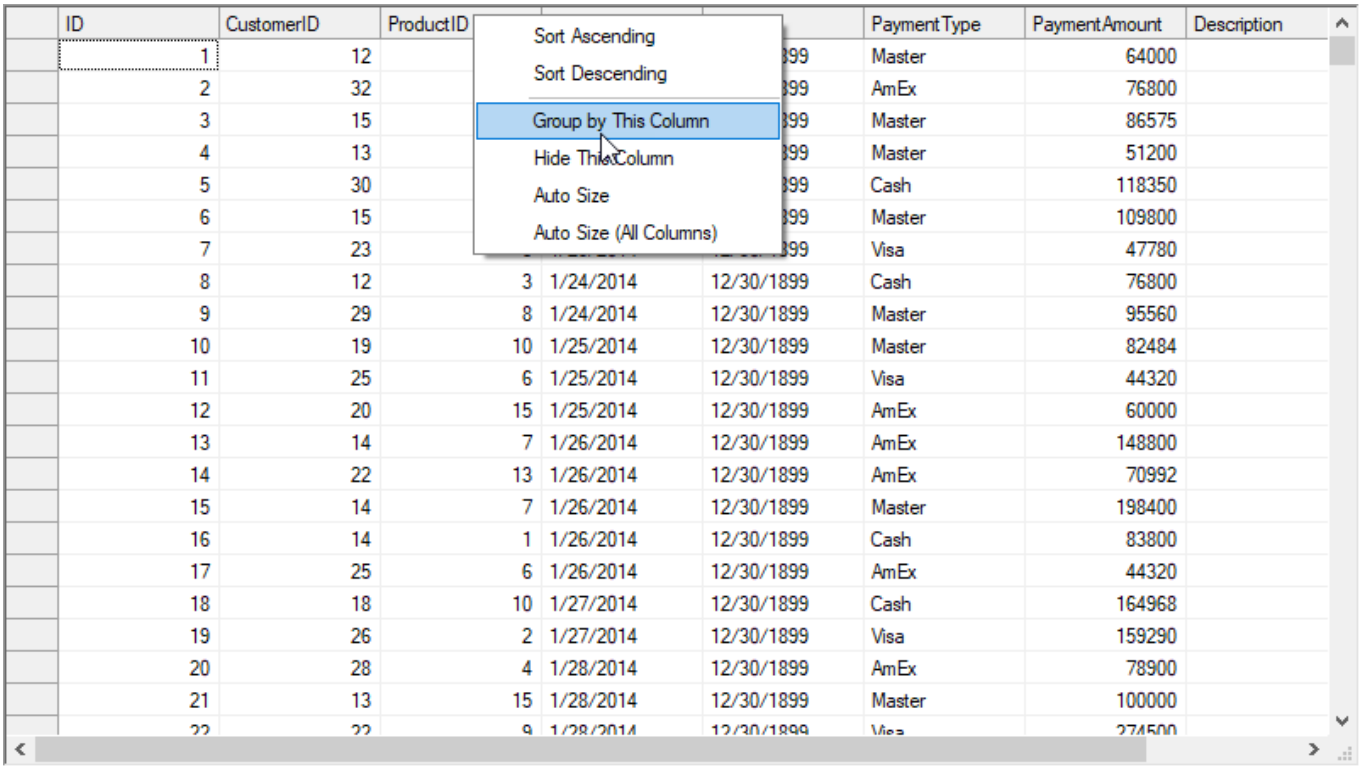
```
// 列と行のサイズを均一に変更できます
c1FlexGrid1.AllowResizing = C1.Win.C1FlexGrid.AllowResizingEnum.Both;

// 最初の列のサイズ変更のみを無効にします
c1FlexGrid1.Cols[1].AllowResizing = false;
```

VB.NET

```
' 列と行のサイズを均一に変更できます
c1FlexGrid1.AllowResizing = C1.Win.C1FlexGrid.AllowResizingEnum.Both

' 最初の列のサイズ変更のみを無効にします
c1FlexGrid1.Cols(1).AllowResizing = False
```

コンテキストメニューの有効化

FlexGrid では、実行時に列操作に関連するアクションをすばやく簡単に実行するために、列コンテキストメニューがサポートされています。右クリックで列コンテキストメニューを使用するには、**C1FlexGrid** クラスにある ColumnContextMenuEnabled プロパティを **true** に設定する必要があります。デフォルトでは、このプロパティは false に設定されています。



ID	CustomerID	ProductID	PaymentType	PaymentAmount	Description
1	12		Master	64000	
2	32		AmEx	76800	
3	15		Master	86575	
4	13		Master	51200	
5	30		Cash	118350	
6	15		Master	109800	
7	23		Visa	47780	
8	12	3 1/24/2014	12/30/1899	Cash	76800
9	29	8 1/24/2014	12/30/1899	Master	95560
10	19	10 1/25/2014	12/30/1899	Master	82484
11	25	6 1/25/2014	12/30/1899	Visa	44320
12	20	15 1/25/2014	12/30/1899	AmEx	60000
13	14	7 1/26/2014	12/30/1899	AmEx	148800
14	22	13 1/26/2014	12/30/1899	AmEx	70992
15	14	7 1/26/2014	12/30/1899	Master	198400
16	14	1 1/26/2014	12/30/1899	Cash	83800
17	25	6 1/26/2014	12/30/1899	AmEx	44320
18	18	10 1/27/2014	12/30/1899	Cash	164968
19	26	2 1/27/2014	12/30/1899	Visa	159290
20	28	4 1/28/2014	12/30/1899	AmEx	78900
21	13	15 1/28/2014	12/30/1899	Master	100000

列のコンテキストメニューには、次のオプションがあります。

オプション	説明
昇順でソート	列を昇順にソートします。
降順でソート	列を降順にソートします。
この列でグループ化	マウスポインタが置かれている列に基づいてグリッドデータをグループ化します。
グループ化解除	グリッドデータのグループ化を解除します。このオプションは、グリッドがグループ化の状態である場合にのみ表示されます。

オプション	説明
この列を非表示にする	マウスポインタが置かれている列を非表示にします。
自動サイズ設定	最長の値に合わせて列のサイズを調整します。
自動サイズ設定（すべての列）	各列の最長の値に合わせて、すべての列のサイズを調整します。

C#

```
// 列のコンテキストメニューを有効にします
c1FlexGrid1.ColumnContextMenuEnabled = true;
```

VB.NET

```
' 列のコンテキストメニューを有効にします
c1FlexGrid1.ColumnContextMenuEnabled = True
```

エディタ

このトピックでは、FlexGrid のさまざまな組み込みエディタと、エディタに関連する操作について説明します。また、カスタムエディタを FlexGrid の列に適用する手順も紹介します。

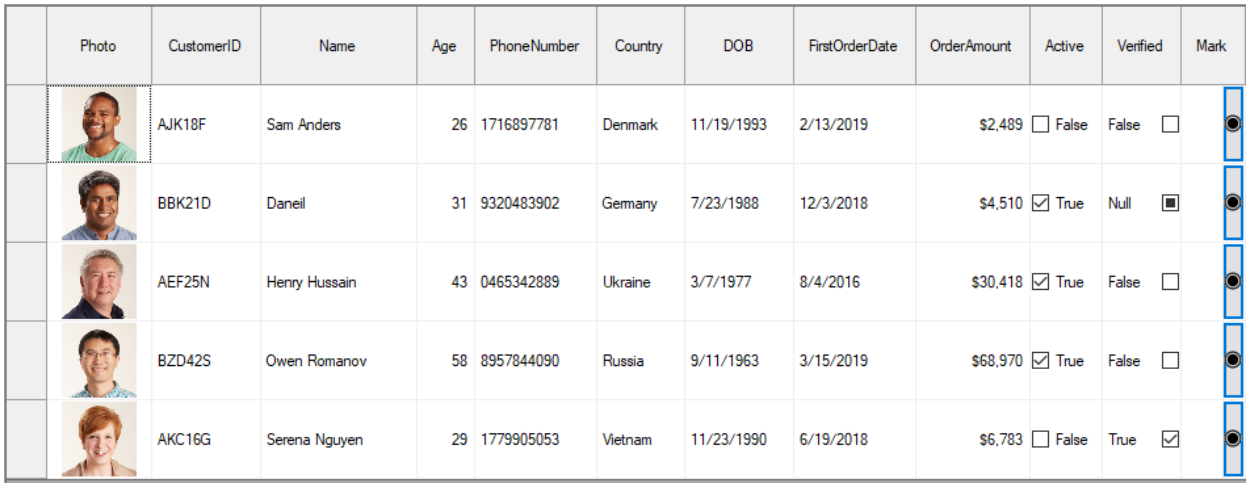
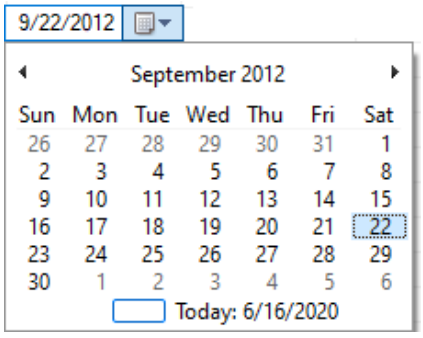
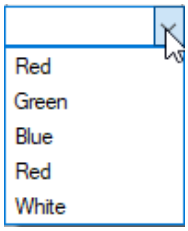


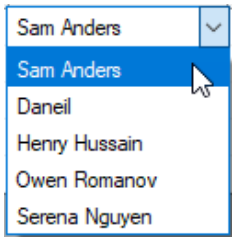
Photo	CustomerID	Name	Age	PhoneNumber	Country	DOB	FirstOrderDate	OrderAmount	Active	Verified	Mark
	AJK18F	Sam Anders	26	1716897781	Denmark	11/19/1993	2/13/2019	$2,489	False	False	
	BBK21D	Daneil	31	9320483902	Germany	7/23/1988	12/3/2018	$4,510	True	Null	
	AEF25N	Henry Hussain	43	0465342889	Ukraine	3/7/1977	8/4/2016	$30,418	True	False	
	BZD42S	Owen Romanov	58	8957844090	Russia	9/11/1963	3/15/2019	$68,970	True	False	
	AKC16G	Serena Nguyen	29	1779905053	Vietnam	11/23/1990	6/19/2018	$6,783	False	True	

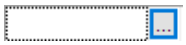
組み込みエディタ

FlexGrid には、セル内の編集を効率的に行えるように、複数の組み込みエディタが用意されています。グリッドは、デフォルトのエディタとして **TextBox** コントロールを使用します。ただし、数値、日付、チェックボックス、コンボボックスなど、他の組み込みエディタもサポートされています。これらのエディタは、通常、列の **DataType** などのいくつかのプロパティの値に基づいて、自動的に割り当てられます。次の表に、簡単な説明とサンプルコードを添えて、FlexGrid が提供する組み込みエディタの概要を示します。組み込みエディタとカスタマイズの詳細については、対応するハイパーリンクをクリックしてください。

組み込みエディタ	スナップショット	説明	サンプルコード
チェックボックス	☑ ☐	**Column** オブジェクトの **DataType** プロパティが **Boolean** に設定されると、自動的に有効になります。	`c1FlexGrid1.Cols[colIndex].DataType = typeof(Boolean);`
数値	17.45 39	**DataType** プロパティが、**Int** や **Decimal** などの数値データ型に設定され	`c1FlexGrid1.Cols[colIndex].DataType = typeof(Int32);`

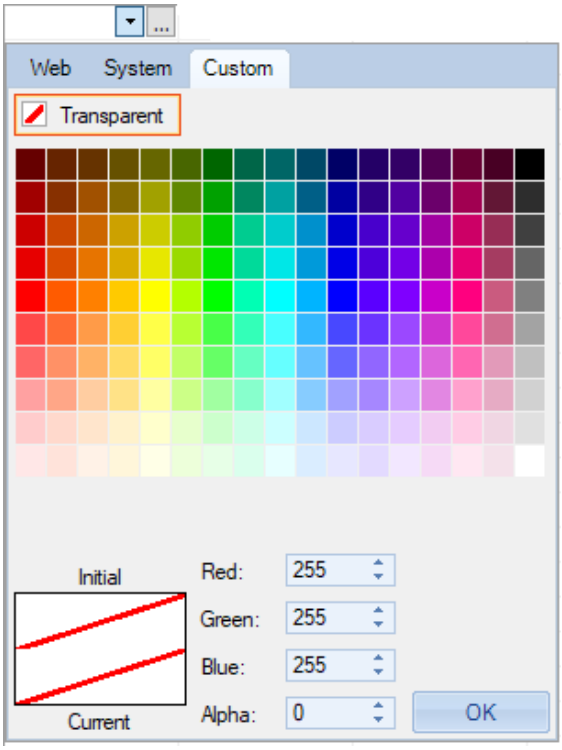
		ると、自動的に有効になります。	
日付		列の **DataType** プロパティが **Date** または **DateTime** に設定されると、自動的に有効になります。	`c1FlexGrid1.Cols[colIndex].DataType = typeof(DateTime);`
ComboBox		**ComboList** プロパティに、複数の値をパイプで区切って設定すると、有効になります。	`c1FlexGrid1.Cols[colIndex].ComboList = "Red|Green|Blue|Red|White";`
マスク	(011) __-____	列の **EditMask** プロパティが設定されると、有効になります。	`c1FlexGrid1.Cols[colIndex].EditMask = "(999) 999-9999";`

マップリスト		**DataMap** プロパティが **IDictionary** オブジェクトに設定されると、有効になります。このオブジェクトは、グリッドに格納されている値とユーザーに表示される値をマップします。	`ListDictionary customerNames = new ListDictionary();`
			`customerNames.Add("AJK18F", "Sam Anders");`
			`customerNames.Add("BBK21D", "Daneil");`
			`customerNames.Add("AEF25N", "Henry Hussain");`
			`customerNames.Add("BZD42S", "Owen Romanov");`
			`customerNames.Add("AKC16G", "Serena Nguyen");`
			
			`c1flexGrid1.Cols["Name"].DataMap = customerNames;`
			`c1flexGrid1.ShowButtons = ShowButtonsEnum.WithFocus;`

セルボタン		**ComboList** プロパティが省略符(...)に設定されると有効になり、編集モードでセルボタンが自動的に表示されます。その後、**CellButtonClick** イベントをキャプチャして、ダイアログボックスを表示するなどの操作を実行できます。	`c1flexGrid1.Cols["colIndex"].ComboList = "...";`
			
			`c1flexGrid1.CellButtonClick += C1flexGrid1_CellButtonClick;`

カスタムエディタ

FlexGrid コントロールは、よく使用される編集オプションのほとんどを上記の組み込みエディタとして提供します。ただし、さらに外部コントロールをエディタとして使用して、特殊な編集ニーズを満たすこともできます。**Control** 基本クラスから派生された任意のコントロールをエディタとして簡単に使用できます。これは、設計時だけでなくコードからも行うことができます。次の例では、**C1ColorPicker** コントロールをセルエディタとして設定しています。



設計時

1. フォームに **C1FlexGrid** コントロールと C1ColorPicker コントロールを追加します。
2. **C1FlexGrid タスク**メニューから**[デザイナ]**オプションを選択して、**C1FlexGrid 列エディタ**を開きます。
3. 列を 1 つ選択し、左側の**[プロパティ]**ペインに切り替えます。
4. **Editor** プロパティに移動して、その値を **C1ColorPicker** コントロールのインスタンスに設定します。

実行時

コードを使用して外部コントロールをエディタとして設定するには、そのコントロールのインスタンスを作成し、それを **Column** オブジェクトの **Editor** プロパティに割り当てます。

外部コントロールを WinForms FlexGrid の列にエディタとして設定する方法については、次のコードを参照してください。

C#

```
// カスタムエディタとして使用するC1ColorPickerコントロールのインスタンスを作成します
C1.Win.C1Input.C1ColorPicker customeditor = new C1.Win.C1Input.C1ColorPicker();

// カスタムエディタにEditorプロパティを割り当てます
c1FlexGrid1.Cols[1].Editor = customeditor;
```

VB.NET

```
' カスタムエディタとして使用するC1ColorPickerコントロールのインスタンスを作成します
Dim customeditor As C1.Win.C1Input.C1ColorPicker = New C1.Win.C1Input.C1ColorPicker()

' カスタムエディタにEditorプロパティを割り当てます
c1FlexGrid1.Cols(1).Editor = customeditor
```

Control 基本クラスから派生していないコントロールでも、**IC1EmbeddedEditor** インタフェースを使用すればエディタとして使用できます。さらに、**UITypeEditor** のクラスをグリッドエディタとして使用することもできます。このように、FlexGrid コントロールは、ほぼすべてのコントロールをセルエディタとして使用できます。実装の詳細については、「**Custom Editors(カスタムエディタ)**」という名前の製品サンプ

ルを参照してください。

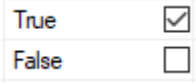
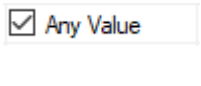
> **メモ: ComponentOneControlPanel.exe** を使用して WinForms Edition をインストールする際にサンプルをインストールした場合、前述の製品サンプルは、システムの \Documents\ComponentOne Samples\WinForms\vx.x.x\C1FlexGrid\CS にあります。

チェックボックス

ブール値へのチェックボックスの表示

FlexGrid では、ブール値を表すためにチェックボックスエディタがデフォルトで使用されます。つまり、Row オブジェクトまたは Column オブジェクトの DataType プロパティが**ブール**型に設定されていると、セルにチェックボックスが表示されます。ユーザーは、このチェックボックスをクリックするだけで編集可能セルの値を切り替えることができます。

このデフォルトの動作を無効にして、ブール型のチェックボックスの代わりにテキスト値を表示するには、**Row** オブジェクトまたは **Column** オブジェクトの Format プロパティを文字列値に設定します。

デフォルトのチェックボックス	テキストとチェックボックス	非ブール値のチェックボックス	3 つの状態を表すチェックボックス
			

WinForms FlexGrid の列にチェックボックスタイプエディタを設定し、それをさらに設定するには、次のコードを使用します。

C#

```
// データ型をブール値に設定します(チェックボックスを自動的に表示します
c1FlexGrid1.Cols["Verified"].DataType = typeof(Boolean);

// チェックボックスを表示する代わりに文字列値を表示します
c1FlexGrid1.Cols["Verified"].Format = "Yes;No";

// ブール列のチェックボックスの横にあるテキストを有効にします
c1FlexGrid1.Cols["Verified"].ImageAndText = true;
```

VB.NET

```
' データ型をブール値に設定します(チェックボックスを自動的に表示します)
c1FlexGrid1.Cols("Verified").DataType = GetType(Boolean)

' チェックボックスを表示する代わりに文字列値を表示します
c1FlexGrid1.Cols("Verified").Format = "Yes;No"

' ブール列のチェックボックスの横にあるテキストを有効にします
c1FlexGrid1.Cols("Verified").ImageAndText = True
```

非ブール値へのチェックボックスの表示

非連結モードでは、チェックボックスと共に、非ブール型のテキストも表示できます。任意の種類のセルにチェックボックスを追加するには、SetCellCheck メソッドを使用します。このメソッドは、行と列のインデックスのほかに、レンダリング時にチェックボックスの状態を指定する CheckEnum 列挙の値をパラメータの 1 つとして受け取ります。

次のコードは、非ブール値にチェックボックスを表示する方法を示しています。

C#

```
// チェックオン状態のセル(3,2)に2つの状態のチェックボックスを設定します
```

```
c1flexGrid1.SetCellCheck(3, 2, CheckEnum.Checked);
```

VB.NET

```
' チェックオン状態のセル(3,2)に2つの状態のチェックボックスを設定します
c1flexGrid1.SetCellCheck(3, 2, CheckEnum.Checked)
```

チェックボックスの配置の設定

セル内のチェックボックスの位置を設定するには、**Row** オブジェクトまたは **Column** オブジェクトの **ImageAlign** プロパティを使用する必要があります。このプロパティは、ImageAlignEnum 列挙に含まれる値を受け取ります。これを使用して、画像を非表示にする、タイル表示する、引き伸ばす、画像の位置を設定するなどの操作が可能です。

チェックボックスを WinForms FlexGrid の列の中央に配置して表示するには、次のコードを使用します。

C#

```
// チェックボックスをセルの右中央に揃えます
c1flexGrid1.Cols["Verified"].ImageAlign = ImageAlignEnum.RightCenter;
```

VB.NET

```
' チェックボックスをセルの右中央に揃えます
c1flexGrid1.Cols("Verified").ImageAlign = ImageAlignEnum.RightCenter
```

チェックボックス画像の変更

さまざまな状態のチェックボックスのアイコン画像を変更するには、Glyphs プロパティからアクセスできる GlyphEnum を使用します。グリフ変更の詳細については、カスタムグリフ を参照してください。

WinForms FlexGrid のさまざまなチェックボックスの状態に使用する画像を変更するには、次のコードを使用します。

C#

```
// カスタム画像をチェックボックスアイコンとして設定します
Image imgChk = new Bitmap("../../Resources/Images/checked.png");
Image imgUnchk = new Bitmap("../../Resources/Images/unchecked.png");
Image imgGray = new Bitmap("../../Resources/Images/null.png");
c1flexGrid1.Glyphs[GlyphEnum.Checked] = imgChk;
c1flexGrid1.Glyphs[GlyphEnum.Unchecked] = imgUnchk;
c1flexGrid1.Glyphs[GlyphEnum.Grayed] = imgGray;
```

VB.NET

```
' カスタム画像をチェックボックスアイコンとして設定します
Dim imgChk As Image = New Bitmap("../../Resources/Images/checked.png")
Dim imgUnchk As Image = New Bitmap("../../Resources/Images/unchecked.png")
Dim imgGray As Image = New Bitmap("../../Resources/Images/null.png")
c1flexGrid1.Glyphs(GlyphEnum.Checked) = imgChk
c1flexGrid1.Glyphs(GlyphEnum.Unchecked) = imgUnchk
c1flexGrid1.Glyphs(GlyphEnum.Grayed) = imgGray
```

3 つの状態を表すチェックボックスの使用

2 つの状態を表す通常のチェックボックスに加えて、FlexGrid では 3 つの状態を表すチェックボックスも作成できます。3 つの状態を有効にする最も簡単な方法は、CheckEnum を使用することです。ブール値チェックボックスでは、**CheckEnum.Checked** と **CheckEnum.Unchecked** の状態が切り替えられるのに対して、3 つの状態を表すチェックボックス

の状態は **CheckEnum.TSChecked**、**CheckEnum.TSUnchecked**、および **CheckEnum.TSGrayed** で表されます。ただし、この場合は、一度チェックボックスの状態をオン/オフに切り替えると、デフォルトでは null に戻すことができません。チェックボックスの 3 つの状態を繰り返し切り替えるには、**ValidateEdit** イベントを処理する必要があります。

WinForms FlexGrid で 3 つの状態を表すチェックボックスを作成するには、次のコードを使用します。

C#

```
    private void c1FlexGrid1_ValidateEdit(object sender, ValidateEditEventArgs e)
         {
       if (c1FlexGrid1.Cols[e.Col].Name == "Done")
        {
          e.Cancel = true;
          if (c1FlexGrid1[e.Row, e.Col].Equals(false))
            {
              c1FlexGrid1[e.Row, e.Col] = true;
            }
          else if (c1FlexGrid1[e.Row, e.Col].Equals(true))
            {
               c1FlexGrid1[e.Row, e.Col] = DBNull.Value;
            }
          else if (c1FlexGrid1[e.Row, e.Col].Equals(DBNull.Value))
            {
               c1FlexGrid1[e.Row, e.Col] = false;
            }
        }
     }
```

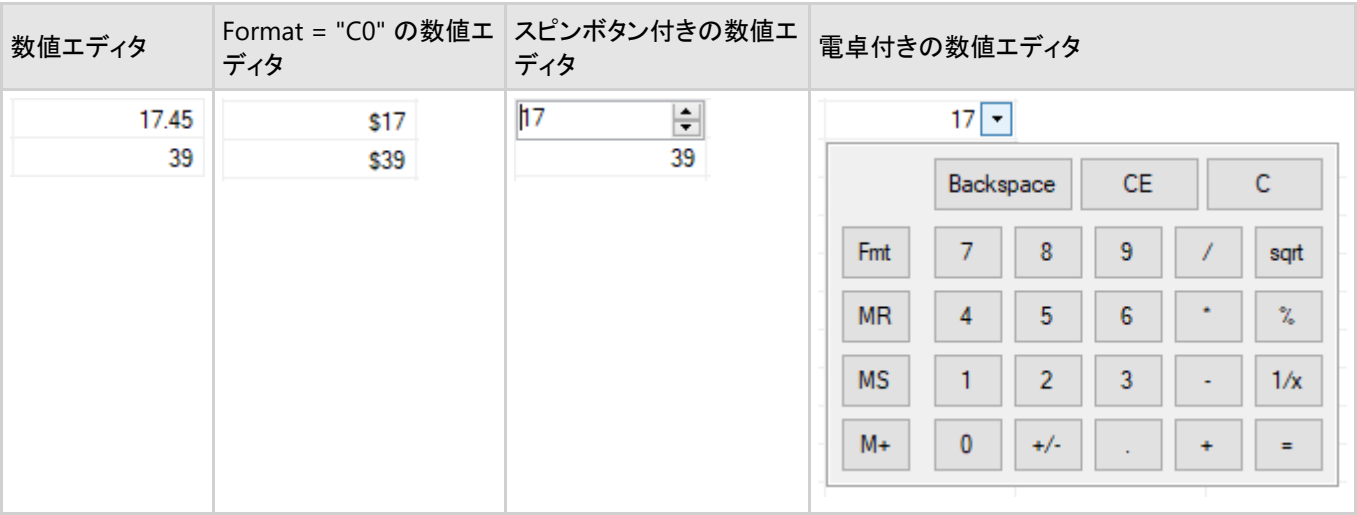
VB.NET

```
Private Sub c1FlexGrid1_ValidateEdit(ByVal sender As Object, ByVal e As
ValidateEditEventArgs)
    If c1FlexGrid1.Cols(e.Col).Name Is "Done" Then
        e.Cancel = True

        If c1FlexGrid1(e.Row, e.Col).Equals(False) Then
            c1FlexGrid1(e.Row, e.Col) = True
        ElseIf c1FlexGrid1(e.Row, e.Col).Equals(True) Then
            c1FlexGrid1(e.Row, e.Col) = DBNull.Value
        ElseIf c1FlexGrid1(e.Row, e.Col).Equals(DBNull.Value) Then
            c1FlexGrid1(e.Row, e.Col) = False
        End If
    End If
End Sub
```

数値

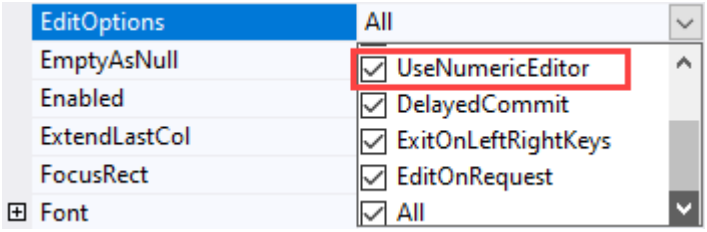
FlexGrid は、数値データを編集するために、デフォルトでは数値エディタを使用します。つまりこれは、データ型が Int や Decimal などの数値型のいずれかに設定されている場合です。この FlexGrid の動作は EditOptions プロパティによって制御されます。このプロパティは、EditFlags 列挙を介してさまざまな編集オプションを提供します。提供されるフラグの 1 つとして **UseNumericEditor** があります。**EditOptions** プロパティのデフォルト値は **All** なので、**UseNumericEditor** フラグは常に ON になります。したがって、数値データの編集時は自動的に数値エディタが有効になります。設計時およびコードから EditOptions プロパティにアクセスできます。

数値エディタ	Format = "C0" の数値エディタ	スピンボタン付きの数値エディタ	電卓付きの数値エディタ
17.45 39	$17 $39	17 39	17



設計時

設計時に数値エディタを有効または無効にするには、**Properties** ウィンドウから EditOptions プロパティにアクセスします。



実行時

実行時は、EditOptions プロパティで UseNumericEditor フラグをオンに設定する必要があります。以下のコードを使用して、WinForms FlexGrid の EditOptions プロパティを設定します。

C#

```
// すべてのフラグをオンに設定します
c1FlexGrid1.EditOptions = C1.Win.C1FlexGrid.EditFlags.All;

// UseNumericEditorフラグのみを設定します
// c1FlexGrid1.EditOptions = C1.Win.C1FlexGrid.EditFlags.UseNumericEditor;
```

VB.NET

```
' すべてのフラグをオンに設定します
c1FlexGrid1.EditOptions = C1.Win.C1FlexGrid.EditFlags.All

' UseNumericEditorフラグのみを設定します
' c1FlexGrid1.EditOptions = C1.Win.C1FlexGrid.EditFlags.UseNumericEditor
```

数値セルの書式設定

数値セル内のデータを書式設定するために、FlexGrid は、Column および Row オブジェクトの Format プロパティを提供します。数値、通貨、指数など、小数値をサポートする書式の場合、FlexGrid はデフォルトで小数点以下 2 位まで表示します。ただし、表示する小数点以下の桁数は、**Format** プロパティの値にその数を付加することで指定できます。たとえば、Number 型のセルに小数点以下 3 位まで値を表示する場合は、Format プロパティの値を「**N3**」または「**n3**」に指定します。

> **メモ**:このプロパティは、保存されている値には影響しません。その値が実行時どのように表示されるかにのみ影響しま

> す。セルの書式設定された値にアクセスするには、C1FlexGrid クラスの **GetDataDisplay(Int32, Int32)** メソッドを使用します。

設計時およびコードから Format プロパティを設定できます。次の表に、数値セルでよく使用される書式をリストします。

書式	値	説明
数値	N または n	単純な数値の書式設定を指定します。
通貨	C または c	金額の書式設定を指定します。
指数表記	E または e	指数表記を使用する書式設定を指定します。
パーセント	P または p	パーセント値の書式設定を指定します。
Custom	ユーザー定義	ユーザーから書式文字列の値を取得します。 カスタム文字列はコードでの処理が必要になる場合があります。

設計時

1. デザインビューで、FlexGrid スマートタグをクリックして **C1FlexGrid タスク**メニューを開きます。
2. **[列タスク]**オプションをクリックし、**Format String** プロパティに移動します。
3. 省略符ボタンをクリックして[書式文字列]ダイアログを開きます。
4. 左のリストボックスから、**[通貨]**などの必要な書式を選択します。
5. ダイアログの右側から、**[小数点以下の桁数]**などの書式固有のプロパティを選択します。

実行時

特定の数値型列の書式を指定するには、**Format** プロパティを上記の値のいずれかに設定します。この例では、列 3 の書式を小数点以下なしの通貨に設定しました。

次のコードは、WinForms FlexGrid の 1 つの列の書式を設定する方法を示します。

C#

```
// 小数点なしの通貨形式を設定します
c1FlexGrid1.Cols[3].Format = "C0";
```

VB.NET

```
' 小数点なしの通貨形式を設定します
c1FlexGrid1.Cols(3).Format = "C0"
```

スピンボタンを使用した入力

入力時にスピンボタンを使用する数値エディタを作成するには、外部コントロールのインスタンスを割り当てる必要があります。たとえば、**NumericUpDown** を **Editor** プロパティに割り当てます。

WinForms FlexGrid でスピンボタンを含む数値エディタを作成するには、次のコードを使用します。

C#

```
// NumericUpDownコントロールを2番目の列のエディタとして割り当てます
c1flexGrid1.Cols[2].Editor = new NumericUpDown();
```

VB.NET

```
' NumericUpDownコントロールを2番目の列のエディタとして割り当てます
c1flexGrid1.Cols(2).Editor = New NumericUpDown()
```

電卓を使用した入力

数値エディタでユーザーが電卓を使用して入力できるようにするには、C1Input ライブラリの C1NumericEdit コントロールをエディタとして使用します。それには、**C1.Win.C1Input** への参照を追加し、**C1NumericEdit** コントロールのインスタンスを **Editor** プロパティに割り当てます。外部コントロールをエディタとして使用する方法については、「カスタムエディタ」を参照してください。

電卓付きの数値エディタを作成するには、次のコードを使用します。

C#

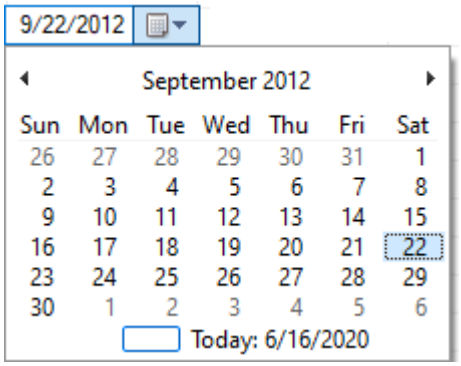
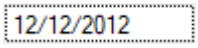
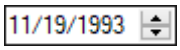
```
// NumericUpDownコントロールを2番目の列のエディタとして割り当てます
c1flexGrid1.Cols[2].Editor = new C1NumericEdit(){ShowUpDownButtons=false};
```

VB.NET

```
' NumericUpDownコントロールを2番目の列のエディタとして割り当てます
c1flexGrid1.Cols(2).Editor = New C1NumericEdit() With
  {
      .ShowUpDownButtons = False
  }
```

日付

FlexGrid は、デフォルトでは、すべての **Date** 型および **DateTime** 型のデータで日付エディタを使用します。日付エディタは、クリックすることでカレンダーが表示されるドロップダウンです。ユーザーは、このカレンダー内を移動して目的の日付を選択できます。ユーザーは、エディタで日、月、年を選択したり上/下矢印キーを押すだけで、日、月、年を変更できます。

デフォルトの日付エディタ	カレンダーなしの日付エディタ	スピンボタン付きの日付エディタ
		

WinForms FlexGrid の列に日付エディタを作成するには、次のコードを使用します。

C#

```
// データ型プロパティをDateTimeに設定します
c1FlexGrid1.Cols[1].DataType = typeof(DateTime);
```

VB.NET

```
' データ型プロパティをDateTimeに設定します
c1FlexGrid1.Cols(1).DataType = GetType(Date)
```

カレンダーを表示しない入力

上記のように、FlexGrid の DateTime 型のセルは、ユーザーからの入力を受け付けるためのドロップダウンカレンダーを自動的に表示します。ただし、カレンダーを表示しないでユーザーから日付入力を受け取ることもできます。それには、EditMask プロパティを使用してマスクを設定し、次に **ValidateEdit** イベントを使用して入力値を検証します。

次のコードは、WinForms FlexGrid でカレンダーなしの日付エディタを作成する方法を示します。

C#

```
// 列にマスクを設定します
c1flexGrid1.Cols[3].EditMask = "00/00/0000";
```

VB.NET

```
' 列にマスクを設定します
c1flexGrid1.Cols(3).EditMask = "00/00/0000"
```

セルのマスクと検証の詳細については、マスク および 検証 のトピックをそれぞれ参照してください。カレンダーを非表示にするもう 1 つの方法は、スピンボタンを使用してユーザー入力を取得することです。

スピンボタンを使用した入力

以下のコードは、スピンボタン付きの日付エディタを作成する方法を示しています。

日付エディタにスピンボタンを表示するには、**SetupEditor** イベントを使用してエディタを DateTimePicker に変換し、その ShowUpDown プロパティを **true** に設定します。

C#

```
    private void C1flexGrid1_SetupEditor(object sender, RowColEventArgs e)
     {
      if (c1flexGrid1.Cols[e.Col].Name == "DOB")
       {
        // FlexGridの現在のセルエディタをキャストします
           var dateEditor = c1flexGrid1.Editor as DateTimePicker;

        // UpDownボタンを表示します(カレンダーのドロップダウンボタンを置き換えます)
           dateEditor.ShowUpDown = true;
       }
     }
```

VB.NET

```
Private Sub C1flexGrid1_SetupEditor(ByVal sender As Object, ByVal e As
RowColEventArgs)
    If c1flexGrid1.Cols(e.Col).Name Is "DOB" Then
        ' FlexGridの現在のセルエディタをキャストします
        Dim dateEditor = TryCast(c1flexGrid1.Editor, DateTimePicker)

        ' UpDownボタンを表示します(カレンダーのドロップダウンボタンを置き換えます)
        dateEditor.ShowUpDown = True
    End If
End Sub
```

日付書式の設定

Date 型の列に書式を設定するには、Column オブジェクトの Format プロパティを設定する必要があります。

次の表に、定義済みの書式をリストします。

書式	値	例
短い日付	d	11/19/2003
長い日付	D	Friday, November 19, 2003
Short Time	t	12:15 AM
Long Time	T	12:15:30 AM

.NET Framework のさまざまな書式指定子を使用して、カスタム書式を作成することもできます。詳細については、Microsoft Web サイトの「カスタム日時書式文字列」を参照してください。

次のコードを使用すると、WinForms FlexGrid 列にカスタム日付書式を作成できます。

C#

```
// 事前定義された書式を設定します
c1FlexGrid1.Cols[DOB].Format = "d";

// カスタム書式を設定します
c1FlexGrid1.Cols[OrderDate].Format = "dd/MM/yyyy";
```

VB.NET

```
' 事前定義された書式を設定します
c1FlexGrid1.Cols(DOB).Format = "d"

' カスタム書式を設定します
c1FlexGrid1.Cols(OrderDate).Format = "dd/MM/yyyy"
```

国固有の日付書式の表示

上記の書式は最もよく使用される日付書式ですが、日本など、固有のカレンダーと日付書式を使用する方がよい場合もあります。固有のカレンダーと日付書式は、C1FlexGrid クラスの **OwnerDrawCell** イベントと System.Globalization 名前空間を使用して表示できます。この名前空間は、カレンダーや日付書式など、カルチャ関連情報を定義するさまざまなクラスを提供します。たとえば、和暦と日本の日付書式を表示するには、System.Globalization.JapaneseCalendar クラスを利用します。同様に、グレゴリオ暦、ヘブライ暦、イスラム暦、韓国暦などのカレンダーも表示できます。

次のコードを使用して、WinForms FlexGrid で国固有の日付書式を設定します。

C#

```
c1flexGrid1.Cols["DOB"].Format = "dd/MM/yyyy";
c1flexGrid1.DrawMode = DrawModeEnum.OwnerDraw;
c1flexGrid1.OwnerDrawCell += C1flexGrid1_OwnerDrawCell;
```

```
    private void C1flexGrid1_OwnerDrawCell(object sender, OwnerDrawCellEventArgs e)
     {
        if (c1flexGrid1.Cols[e.Col].DataType == typeof(DateTime) && e.Row >=
c1flexGrid1.Rows.Fixed)
          {
            e.Text = DateTime.Parse(e.Text).ToString("yyyy年MM月dd日(dddd)", c);
          }
     }
```

VB.NET

```
 C1FlexGrid1.Cols("DOB").Format = "dd/MM/yyyy"

C1FlexGrid1.DrawMode = DrawModeEnum.OwnerDraw

AddHandler C1FlexGrid1.OwnerDrawCell, AddressOf C1flexGrid1_OwnerDrawCell

Private Sub C1flexGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e As
OwnerDrawCellEventArgs)
     If C1FlexGrid1.Cols(e.Col).DataType = GetType(DateTime) AndAlso e.Row >=
C1FlexGrid1.Rows.Fixed Then
        e.Text = DateTime.Parse(e.Text).ToString("yyyy年MM月dd日(dddd)", c)
     End If
 End Sub
```

最小/最大日付の設定

有効な値の範囲を設定するには、DateTimePicker コントロールをエディタとして使用し、その MinDate プロパティと MaxDate プロパティを設定します。

WinForms FlexGrid で有効な日付の範囲を設定するには、次のコードを使用します。

C#

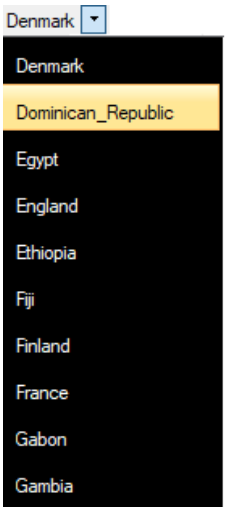
```
    DateTimePicker dateTimePicker = new DateTimePicker();
    dateTimePicker.Format = DateTimePickerFormat.Short;

    // 最大日と最小日を設定します
    dateTimePicker.MinDate = new DateTime(2015, 05, 01);
    dateTimePicker.MaxDate = DateTime.Today;

    // DateTimePickerコントロールをFirstOrderDate(date)列のエディターとして割り当てます
    c1flexGrid1.Cols["FirstOrderDate"].Editor = dateTimePicker;
```

VB.NET

```
     Dim dateTimePicker As DateTimePicker = New DateTimePicker()
     dateTimePicker.Format = DateTimePickerFormat.[Short]

     ' 最大日と最小日を設定します
     dateTimePicker.MinDate = New DateTime(2015, 05, 01)
     dateTimePicker.MaxDate = Date.Today

     ' DateTimePickerコントロールをFirstOrderDate(date)列のエディターとして割り当てます
     c1flexGrid1.Cols("FirstOrderDate").Editor = dateTimePicker
```

コンボボックス

コンボボックスセルは、使用可能な値が明確に定義されたリストをドロップダウンリストの形式でユーザーに提供するためによく使用されます。FlexGrid では、この複数オプションエディタが、ドロップダウンリスト、ドロップダウンコンボ、省略符ボタン、テキストボックスと省略符ボタンなどのさまざまな形式で提供されます。

デフォルトのコンボボックス	複数列のコンボボックス	カスタム背景色のコンボボックス	画像を含むコンボボックス
			

ドロップダウンリストまたはドロップダウンコンボの表示

FlexGrid でドロップダウンリストを作成するには、すべての選択肢をパイプ文字で区切った文字列を作成し(「True|False|Don't know」など)、それを Row オブジェクトまたは Column オブジェクトの ComboList プロパティに割り当てます。これで、グリッドのセルの横にドロップダウンボタンが表示されます。ユーザーはドロップダウンボタンをクリックするか**[F2]**キーを押して、そのセルで使用できる選択肢のリストを表示できます。

ほかにもよくある状況として、セルによく使用される値のリストを表示するが、ユーザーにもカスタム値の入力を許可する場合があります。この場合は、テキストボックスとドロップダウンリストを組み合わせたドロップダウンコンボを使用します。FlexGrid でコンボを作成するには、最初にパイプ文字で区切られた選択肢のリスト(「|True|False|Don't know」など)を作成し、それを **ComboList** プロパティに割り当てます。

次のコードは、WinForms FlexGrid でドロップダウンリストまたはコンボエディタを作成する方法を示しています。

C#

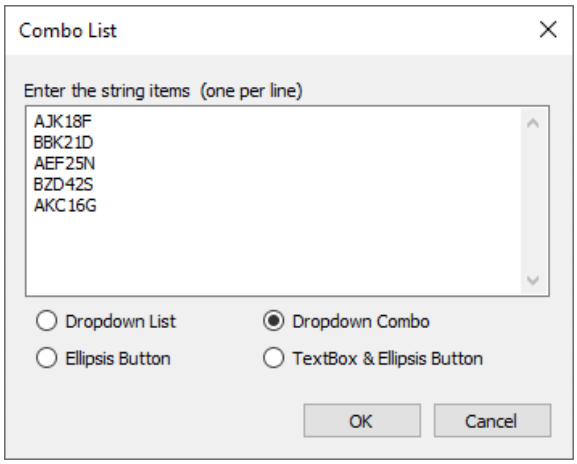
```
// ドロップダウンリストとしてCustomerIDを割り当てます
c1flexGrid1.Cols["CustomerID"].ComboList = "AJK18F|BBK21D|AEF25N|BZD42S|AKC16G";

// ドロップダウンコンボボックスとしてCustomerIDを割り当て、パイプ文字"|"でリストを開始します
// c1flexGrid1.Cols["CustomerID"].ComboList = "|AJK18F|BBK21D|AEF25N|BZD42S|AKC16G";
```

VB.NET

```
' ドロップダウンリストとしてCustomerIDを割り当てます
c1flexGrid1.Cols("CustomerID").ComboList = "AJK18F|BBK21D|AEF25N|BZD42S|AKC16G"

' ドロップダウンコンボボックスとしてCustomerIDを割り当て、パイプ文字"|"でリストを開始します
' c1flexGrid1.Cols("CustomerID").ComboList = "|AJK18F|BBK21D|AEF25N|BZD42S|AKC16G"
```

ComboList プロパティは、設計時に**[コンボリスト]**ダイアログを使用して設定することもできます。[コンボリスト]ダイアログにアクセスするには、次の手順に従います。

1. エディタを設定する列の**列タスク**メニューを開きます。
2. **[コンボリスト]**オプションに移動して、フィールドの右側にある省略符ボタンをクリックします。
3. [コンボリスト]ダイアログが開きます。このダイアログで、新しい行にオプション値を 1 つずつ指定できます。
4. これらの値を**ドロップダウンリスト**として表示するか、**ドロップダウンコンボ**として表示するかを選択することもできます。**[省略符ボタン]**オプションまたは**[テキストボックスと省略符ボタン]**オプションを使用して、セルボタンを作成することもできます。



複数列コンボボックスの表示

FlexGrid では、MultiColumnDictionary クラスを使用して、コンボボックスに複数の列を表示することもできます。このクラスは IC1MultiColumnDictionary インタフェースを実装し、複数のオーバーロードがあります。このオーバーロードを使用して、データソースオブジェクト、キー列、複数の列に表示する列、およびコンボボックスを閉じたときに表示する列を指定できます。

WinForms FlexGrid の列に複数列のコンボボックスを表示するには、次のコードを使用します。

C#

```
string[] columnRange = new string[] { "City", "Country" };
c1FlexGrid1.Cols["City"].DataMap = new MultiColumnDictionary(dt, "City", columnRange, 0);
```

VB.NET

```
Dim columnRange = New String() {"City", "Country"}
c1FlexGrid1.Cols("City").DataMap = New MultiColumnDictionary(dt, "City", columnRange, 0)
```

このコードでは、データソースオブジェクト **dt** を使用してグリッドにデータを提供しています。要件に基づいてデータソースを設定できます。

C#

```
    DataTable dt = new DataTable();
    dt.Columns.Add("CustomerID", typeof(int));
    dt.Columns.Add("ContactName", typeof(string));
    dt.Columns.Add("Designation", typeof(string));
    dt.Columns.Add("City", typeof(string));
    dt.Columns.Add("Country", typeof(object));

// サンプルデータ
    dt.Rows.Add(1, "Maria Anders", "Sales Representative", "Madrid", "Spain" );
    dt.Rows.Add(2, "Ana Trujillo", "Sales Associate", "Monterrey", "Mexico" );
    dt.Rows.Add(3, "Antonio Moreno", "Owner", "Dublin", "Ireland");
    dt.Rows.Add(4, "Thomas Hardy", "Sales Representative", "Bristol", "UK");
    dt.Rows.Add(5, "Patricio Simpson", "Marketing Manager", "Munich", "Germany");
    dt.Rows.Add(6, "Paolo Accorti", "Sales Representative", "Barcelona", "Spain");
    dt.Rows.Add(7, "Martine Rancé", "Owner", "Puebla", "Mexico");
    dt.Rows.Add(8, "Elizabeth Brown", "Marketing Manager", "London", "UK");
    dt.Rows.Add(9, "Jaime Yorres", "Order Administrator", "Vienna", "Austria");
    dt.Rows.Add(10, "Yvonne Moncada", "Marketing Manager", "Mexico", "Mexico");
    dt.Rows.Add(11, "Helen Bennett", "Owner/Marketing", "Cork", "Ireland");
    dt.Rows.Add(12, "Sergio Gutiérrezy", "Order Administrator", "Sao Paulo", "Brazil");

    c1FlexGrid1.DataSource = dt;
```

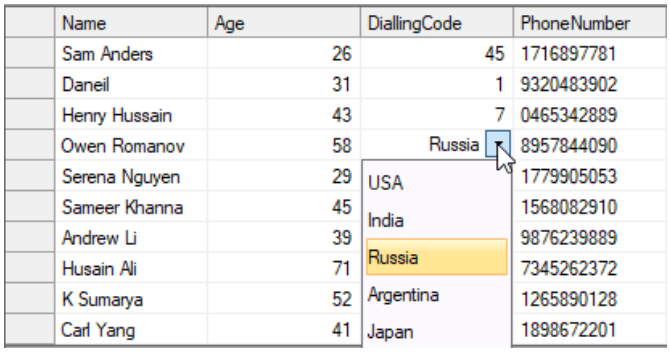
VB.NET

```
Dim dt As DataTable = New DataTable()
dt.Columns.Add("CustomerID", GetType(Integer))
dt.Columns.Add("ContactName", GetType(String))
dt.Columns.Add("Designation", GetType(String))
dt.Columns.Add("Country", GetType(Object))
dt.Columns.Add("City", GetType(String))

' サンプルデータ
dt.Rows.Add(1, "Maria Anders", "Sales Representative", "Spain", "Madrid")
```

```
dt.Rows.Add(2, "Ana Trujillo", "Sales Associate", "Mexico", "Monterrey")
dt.Rows.Add(3, "Antonio Moreno", "Owner", "Ireland", "Dublin")
dt.Rows.Add(4, "Thomas Hardy", "Sales Representative", "UK", "Bristol")
dt.Rows.Add(5, "Patricio Simpson", "Marketing Manager", "Germany", "Munich")
dt.Rows.Add(6, "Paolo Accorti", "Sales Representative", "Spain", "Barcelona")
dt.Rows.Add(7, "Martine Rancé", "Owner", "Mexico", "Puebla")
dt.Rows.Add(8, "Elizabeth Brown", "Marketing Manager", "UK", "London")
dt.Rows.Add(9, "Jaime Yorres", "Order Administrator", "Austria", "Vienna")
dt.Rows.Add(10, "Yvonne Moncada", "Marketing Manager", "Mexico", "Mexico")
dt.Rows.Add(11, "Helen Bennett", "Owner/Marketing", "Ireland", "Cork")
dt.Rows.Add(12, "Sergio Gutiérrezy", "Order Administrator", "Brazil", "Sao Paulo")
c1FlexGrid1.DataSource = dt
```

コンボボックスへのマップデータの設定

コンボボックスに実際の値と異なる値を表示するように設定するには、C1ComboBox をエディタとして使用し、その ItemsDisplayMember プロパティと ItemsValueMember プロパティを利用する必要があります。たとえば、次の例では、実際の値はそれぞれの国の国番号ですが、表示する値としては国名を使用しています。



次のコードは、WinForms FlexGrid のコンボボックス列にマップデータを設定する方法を示しています。

C#

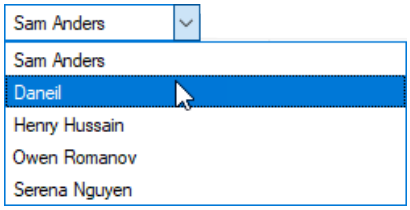
```
private void Form1_Load(object sender, EventArgs e)
{
 // 顧客データを取得します
 Customers = GetCustomers();

 // FlexGridにデータを入力します
 c1FlexGrid1.DataSource = Customers;

 // 国とその国コードのコレクションを作成します
 ObservableCollection<Country> countries = new ObservableCollection<Country>();
 countries.Add(new Country(1, "USA"));
 countries.Add(new Country(91, "India"));
 countries.Add(new Country(7, "Russia"));
 countries.Add(new Country(54, "Argentina"));
 countries.Add(new Country(81, "Japan"));
 countries.Add(new Country(380, "Ukraine"));
 countries.Add(new Country(98, "Iran"));
 countries.Add(new Country(45, "Denmark"));
 countries.Add(new Country(84, "Vietnam"));
 countries.Add(new Country(49, "Germany"));
 BindingSource countryBS = new BindingSource();
 countryBS.DataSource = countries;

 // C1Comboboxコントロールをインスタンス化して設定します
 C1ComboBox countryCodeCombo = new C1ComboBox();
 countryCodeCombo.ItemsDataSource = countryBS;

 #region Mapped Data using C1Combobox

 // 国コードと国名をマッピングするためにプロパティを設定します
 countryCodeCombo.ItemsDisplayMember = "CountryName";
 countryCodeCombo.ItemsValueMember = "CountryCode";
 countryCodeCombo.TextDetached = true;
 countryCodeCombo.TranslateValue = true;

 // C1ComboboxをDiallingCode列のエディタとして設定します
 c1FlexGrid1.Cols["DiallingCode"].Editor = countryCodeCombo;
 #endregion
}
```

VB.NET

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' 顧客データを取得します
    Customers = GetCustomers()

    ' FlexGridにデータを入力します
    c1FlexGrid1.DataSource = Customers

    ' 国とその国コードのコレクションを作成します
    Dim countries As ObservableCollection(Of Country) = New ObservableCollection(Of Country)()
    countries.Add(New Country(1, "USA"))
    countries.Add(New Country(91, "India"))
    countries.Add(New Country(7, "Russia"))
    countries.Add(New Country(54, "Argentina"))
    countries.Add(New Country(81, "Japan"))
    countries.Add(New Country(380, "Ukraine"))
    countries.Add(New Country(98, "Iran"))
    countries.Add(New Country(45, "Denmark"))
    countries.Add(New Country(84, "Vietnam"))
    countries.Add(New Country(49, "Germany"))
    Dim countryBS As BindingSource = New BindingSource()
    countryBS.DataSource = countries

    ' C1Comboboxコントロールをインスタンス化して設定します
    Dim countryCodeCombo As C1ComboBox = New C1ComboBox()
    countryCodeCombo.ItemsDataSource = countryBS

    ' 国コードと国名をマッピングするためにプロパティを設定します
    countryCodeCombo.ItemsDisplayMember = "CountryName"
    countryCodeCombo.ItemsValueMember = "CountryCode"
    countryCodeCombo.TextDetached = True
    countryCodeCombo.TranslateValue = True

    ' C1ComboboxをDiallingCode列のエディタとして設定します
    c1FlexGrid1.Cols("DiallingCode").Editor = countryCodeCombo
End Sub
```

コンボボックスのサイズの設定

コンボボックスのドロップダウンの高さと幅を設定するには、**ComboBox** のインスタンスをエディタとして割り当て、そのインスタンスの DropDownHeight プロパティおよび DropDownWidth プロパティを設定する必要があります。



WinForms FlexGrid に表示するコンボボックスの高さと幅を指定するには、次のコードを使用します。

C#

```
    ComboBox cb = (ComboBox)c1flexGrid1.Editor;
    cb.DropDownWidth = 250;
    cb.DropDownHeight = 200;
```

VB.NET

```
    Dim cb As ComboBox = CType(c1flexGrid1.Editor, ComboBox)
    cb.DropDownWidth = 250
    cb.DropDownHeight = 200
```

背景色やフォントの色の変更

コンボボックスリストの背景色やフォントの色を変更するには、**ComboBox** のインスタンスを作成し、それをエディタとして割り当てます。次に、そのインスタンスの BackColor プロパティおよび ForeColor プロパティを設定します。

WinForms FlexGrid のコンボボックスをカスタマイズするには、次のコードを使用します。

C#

```
ComboBox comboBox = (ComboBox)c1flexGrid1.Editor;
comboBox.BackColor = Color.Black;
comboBox.ForeColor = Color.White;
```

VB.NET

```
Dim comboBox As ComboBox = CType(c1flexGrid1.Editor, ComboBox)
comboBox.BackColor = Color.Black
comboBox.ForeColor = Color.White
```

リストへの画像の表示

コンボボックスリストに画像を表示するには、**C1ComboBox** をエディタとして使用し、その ItemsImageList プロパティを利用する必要があります。このプロパティは ImageList クラス型です。このクラスは、その **Images** コレクションに保存される画像のコンテナになります。最初に、ResourceManager.GetResourceSet メソッドを使用してプロジェクトリソースに保存されている画像にアクセスし、画像とそれに対応する名前をマップするためのコレクションを作成します。このコレクションが ComboBox のデータソースになります。次に、**ImageList** クラスのインスタンスを作成し、データソースから画像をその **Images** コレクションに追加します。この ImageList クラスのインスタンスを **ItemsImageList** プロパティに割り当てて、コンボボックスリストに画像をレンダリングします。

次のコードは、WinForms FlexGrid のコンボボックスリストに画像を表示する方法を示しています。

C#

```
// コンボボックスに画像を入力します
    var itemsSource = new List<Flag>();
    ImageList imgFlag = new ImageList();
    imgFlag.Images.Clear();
    var rsrSet = Resources.ResourceManager.GetResourceSet(CultureInfo.CurrentCulture, true, true);

 foreach(DictionaryEntry entry in rsrSet)
 {
    var img = entry.Value as Image;
    itemsSource.Add(new Flag(entry.Key.ToString(), img));
    }
 itemsSource.Sort(new CompareFlag());

  foreach (Flag entry in itemsSource)
 {
     imgFlag.Images.Add(entry.CountryName, entry.CountryFlag);
  }

 countryCombo.ItemsImageList = imgFlag;
 countryCombo.ItemMode = ComboItemMode.HtmlPattern;
 countryCombo.HtmlPattern = "<div <div style='padding:0px'><img src='{CountryName}' style='padding-
right:14px'>{CountryName}</div>";
```

VB.NET

```
' コンボボックスに画像を入力します
Dim itemsSource = New List(Of Flag)()
Dim imgFlag As ImageList = New ImageList()
imgFlag.Images.Clear()
Dim rsrSet = Resources.ResourceManager.GetResourceSet(CultureInfo.CurrentCulture, True, True)

For Each entry As DictionaryEntry In rsrSet
    Dim img = TryCast(entry.Value, Image)
    itemsSource.Add(New Flag(entry.Key.ToString(), img))
Next

itemsSource.Sort(New CompareFlag())

For Each entry In itemsSource
    imgFlag.Images.Add(entry.CountryName, entry.CountryFlag)
Next

countryCombo.ItemsImageList = imgFlag
countryCombo.ItemMode = ComboItemMode.HtmlPattern
countryCombo.HtmlPattern = "<div <div style='padding:0px'><img src='{CountryName}' style='padding-
right:14px'>{CountryName}</div>"
```

表示する項目の数の設定

コンボボックスに表示する項目の数を設定するには、ComboBox.MaxDropDownItems プロパティを使用します。

WinForms FlexGrid のコンボボックスリストに表示する項目の数を制限するには、次のコードを使用します。

C#

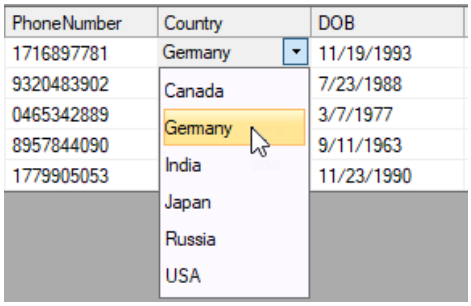
```
// コンボボックスのドロップダウンに表示する国の数を設定します
   countryCombo.MaxDropDownItems = 10;
```

VB.NET

```
' コンボボックスのドロップダウンに表示する国の数を設定します
countryCombo.MaxDropDownItems = 10
```

コンボボックスリストのソート

コンボボックスドロップダウンリストの項目をソートするには、**C1ComboBox** をエディタとして使用し、**Sort** メソッドを呼び出して、コンボボックスに表示されるドロップダウン項目をソートします。



WinForms FlexGrid のコンボボックスリストに項目をソートして表示するには、次のコードを使用します。

C#

```
C1ComboBox countryCombo = new C1ComboBox();
countryCombo.DropDownStyle = DropDownStyle.DropDownList;
List<string> countries = new List<string> { "USA", "Canada", "India", "Russia", "Japan", "Germany" };
countries.Sort();
countryCombo.ItemsDataSource = countries;

// C1ComboboxをCountry列のエディタとして設定します
c1FlexGrid1.Cols["Country"].Editor = countryCombo;
```

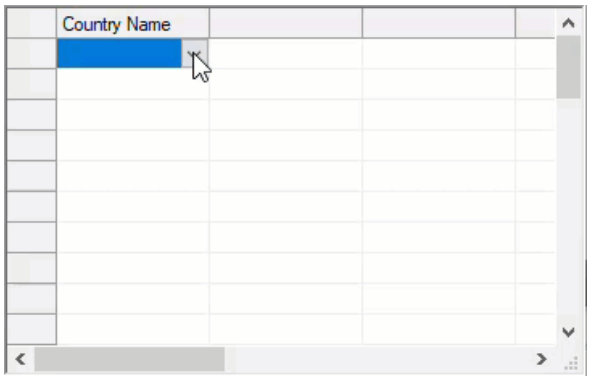
VB.NET

```
Dim countryCombo As C1ComboBox = New C1ComboBox()
countryCombo.DropDownStyle = DropDownStyle.DropDownList
Dim countries As List(Of String) = New List(Of String) From {
    "USA",
    "Canada",
    "India",
    "Russia",
    "Japan",
    "Germany"
}
countries.Sort()
countryCombo.ItemsDataSource = countries

' C1ComboboxをCountry列のエディタとして設定します
c1FlexGrid1.Cols("Country").Editor = countryCombo
```

選択インデックスの取得

選択インデックスまたは選択項目の値を取得するには、ComboBoxEditor クラスの **SelectedIndex** プロパティまたは **SelectedItem** プロパティを使用します。次の例では、**ComboCloseUp** イベントを使用して、選択インデックスと選択項目を示すメッセージボックスを表示しています。

WinForms FlexGrid のコンボボックスリストで選択項目のインデックスと値を取得するには、次のコードを使用します。



C#

```
private void C1FlexGrid1_ComboCloseUp(object sender, C1.Win.C1FlexGrid.RowColEventArgs e)
{
  MessageBox.Show("Selected Index:" +
  c1FlexGrid1.ComboBoxEditor.SelectedIndex + "\n" +
  "Selected Item:" +
  c1FlexGrid1.ComboBoxEditor.SelectedItem);
}
```

VB.NET

```
Private Sub C1FlexGrid1_ComboCloseUp(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs)
    MessageBox.Show("Selected Index:" & c1FlexGrid1.ComboBoxEditor.SelectedIndex & vbLf & "Selected Item:"
+ c1FlexGrid1.ComboBoxEditor.SelectedItem)
End Sub
```

マスク

マスクエディタとは、ユーザー入力を取得するために使用され、ユーザーが入力すると同時にその文字列を自動的に検証する、事前定義されたテンプレートです。マスク文字列には、リテラル文字とテンプレート文字の 2 つの種類があります。リテラル文字は、入力の一部となる文字です。一方、テンプレート文字は、特定のカテゴリ(数字、アルファベットなど)に属する文字のプレースホルダになります。たとえば、次のコードでは、電話番号を格納する最初の列に編集マスク「(999) 999-9999」が割り当てられます。ここで、かっこ「()」やハイフン「-」などの特殊文字はリテラル文字で、数字「9」は任意の数字を表すプレースホルダです。

(011)___-____

FlexGrid コントロールでは、EditMask プロパティによってマスク付き編集をサポートします。これは、通常のテキストフィールドおよびドロップダウンコンボフィールドで使用できます。同じ列内で複数のマスクを適用するには、BeforEdit イベントをトラップして、**EditMask** プロパティを適切な値に設定します。

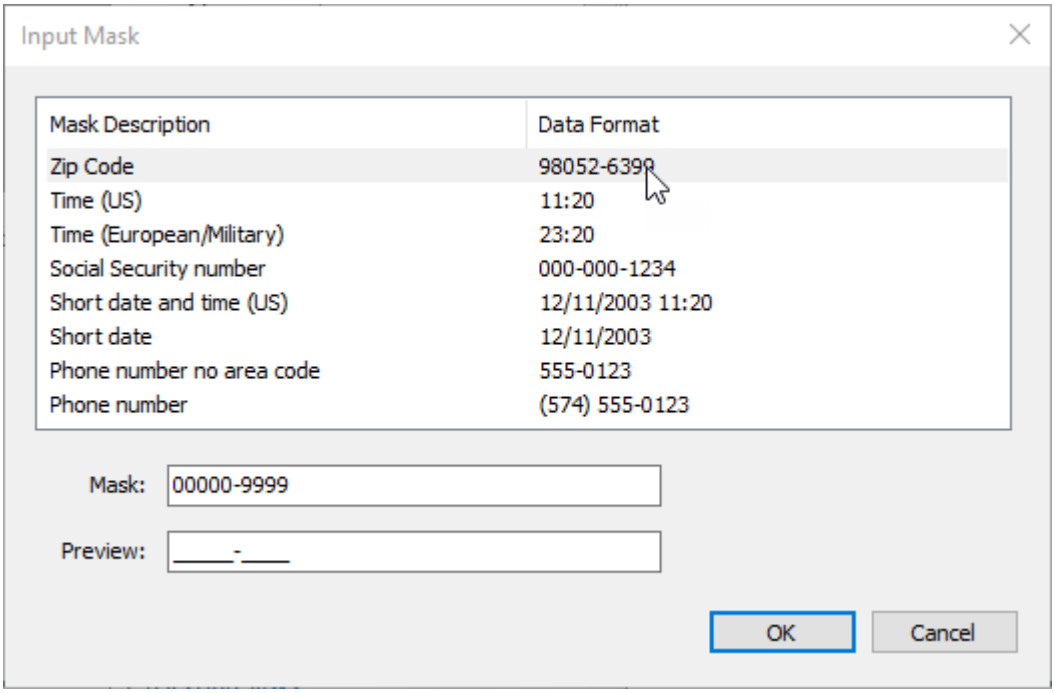
C#

```
// 電話番号の編集マスクを設定します
c1FlexGrid1.Cols[1].EditMask = "(999) 999-9999";
```

VB.NET

```
' 電話番号の編集マスクを設定します
c1FlexGrid1.Cols(1).EditMask = "(999) 999-9999"
```

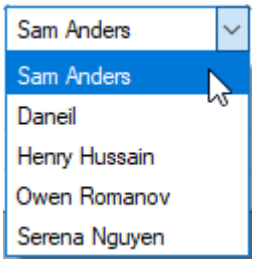
また、**[入力マスク]**ダイアログを使用して、設計時に **EditMask** プロパティを設定することもできます。このダイアログにアクセスするには、**列タスク**メニューの**[編集マスク]**フィールドにある省略符ボタンをクリックします。つまり、この[編集マスク]フィールドは、この時点で選択されている列に固有になります。



EditMask プロパティが空以外の文字列に設定されている場合は、列に **DateTime** 値が含まれていても、マスクエディタが使用されます。通常、DateTime 型のセルの編集には、組み込み日付エディタが使用されます。

マップリスト

マップリストは、グリッドに保存されているデータをユーザーが理解できる値にマップします。このようなマッピングは、実際にグリッドに保存されているデータがエンコードされていたり、ユーザーによる理解が難しい場合に、ユーザーフレンドリな値を表示するためによく使用されます。たとえば、従業員レコードのテーブルの場合、データベースには従業員 ID が保存されていても、管理職ユーザーには従業員の名前が表示されないと不便です。



データマッピングの表示

FlexGrid でデータマッピングを実行するには、DataMap プロパティを使用します。このプロパティには、データベース値とグリッドに表示される値をマップする **IDictionary** オブジェクトへの参照が含まれます。**HashTable**、**ListDictionary**、および **SortedList** は、有効なデータマップを提供する **IDictionary** オブジェクトの一例です。

次のコードは、WinForms FlexGrid でデータマップリストを表示する方法を示しています。

C#

```
// データマップを作成します
ListDictionary customerNames = new ListDictionary();
customerNames.Add("AJK18F", "Sam Anders");
customerNames.Add("BBK21D", "Daneil");
customerNames.Add("AEF25N", "Henry Hussain");
customerNames.Add("BZD42S", "Owen Romanov");
customerNames.Add("AKC16G", "Serena Nguyen");
```

```
// データマップをflexgrid列に割り当てます
c1flexGrid1.Cols["Name"].DataMap = customerNames;
```

VB.NET

```
' データマップを作成します
Dim customerNames As ListDictionary = New ListDictionary()
customerNames.Add("AJK18F", "Sam Anders")
customerNames.Add("BBK21D", "Daneil")
customerNames.Add("AEF25N", "Henry Hussain")
customerNames.Add("BZD42S", "Owen Romanov")
customerNames.Add("AKC16G", "Serena Nguyen")

' データマップをflexgrid列に割り当てます
c1flexGrid1.Cols("Name").DataMap = customerNames
```

HashTable、ListDictionary、SortedList の各クラスでは、項目の並び順が異なります。このため、編集可能な列でこれらのテーブルを使用する場合は、マップリストにレンダリングされる項目の順序も、リストの作成に使用されるキーとクラスによって異なります。

データマップのキーの型は、編集対象のセルの型と同じである必要があります。たとえば、列に短整数（Int16）が含まれている場合、その列に関連するデータマップには単整数のキーが含まれている必要があります。

イメージマップの表示

FlexGrid 列にイメージマップを表示するには、Row オブジェクトまたは Column オブジェクトの **ImageMap** プロパティを、画像と対応するテキスト値をマップする **IDictionary** オブジェクトに設定する必要があります。たとえば、列に国名が格納されている場合は、このプロパティを使用して、対応する国旗を表示できます。オブジェクトの ImageAndText プロパティを使用して、画像のみを表示するか、画像とテキストを表示するかを制御できます。

WinForms FlexGrid でイメージマップを作成するには、次のコードを使用します。

C#

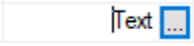
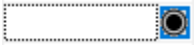
```
Hashtable ht = new Hashtable();
foreach (Row row in c1flexGrid1.Rows)
{
    ht.Add(row["CustomerID"], LoadImage(row["Photo"] as byte[]));
}

// ImageMapをPhoto列に割り当てます
c1flexGrid1.Cols["Photo"].ImageMap = ht;

// アスペクト比を維持しながら、画像をセルに合わせます
c1flexGrid1.Cols["Photo"].ImageAlign = ImageAlignEnum.Scale;
c1flexGrid1.Cols["Photo"].Width = 80;
```

VB.NET

```
    Dim ht As Hashtable = New Hashtable()

    For Each row As Row In c1flexGrid1.Rows
        ht.Add(row("CustomerID"), LoadImage(TryCast(row("Photo"), Byte())))
    Next

    ' ImageMapをPhoto列に割り当てます
    c1flexGrid1.Cols("Photo").ImageMap = ht

    ' アスペクト比を維持しながら、画像をセルに合わせます
    c1flexGrid1.Cols("Photo").ImageAlign = ImageAlignEnum.Scale
    c1flexGrid1.Cols("Photo").Width = 80
```

セルボタン

セルボタンエディタとは、省略符ボタンを含むセルのことです。1 回のボタンクリックで操作を完了したりダイアログを開く必要がある場合に、このエディタを使用します。通常、そのようなダイアログには、ユーザーが選択できる複数のオプションや設定が含まれています。

デフォルトのセルボタン	セルボタンとテキスト	カスタム画像を含むセルボタン
		

セルボタンの表示

FlexGrid でセルボタンを表示するには、ComboList プロパティを省略符「...」に設定します。また、省略符ボタンの前にパイプ文字を使用することで、ユーザーが文字を入力できるテキストボックスを省略符と共に表示できます。ユーザーがセルボタンをクリックすると、**CellButtonClick** イベントが発生します。このイベントをキャプチャして、必要な操作を実装したり、目的のダイアログを表示することができます。

デフォルトでは、セルが編集モードになったときにセルボタンが表示されます。ただし、ShowButtons プロパティを **Always** に設定すると、非編集モードでもセルボタンが表示されます。

WinForms FlexGrid の列にセルボタンを表示するには、次のコードを使用します。

C#

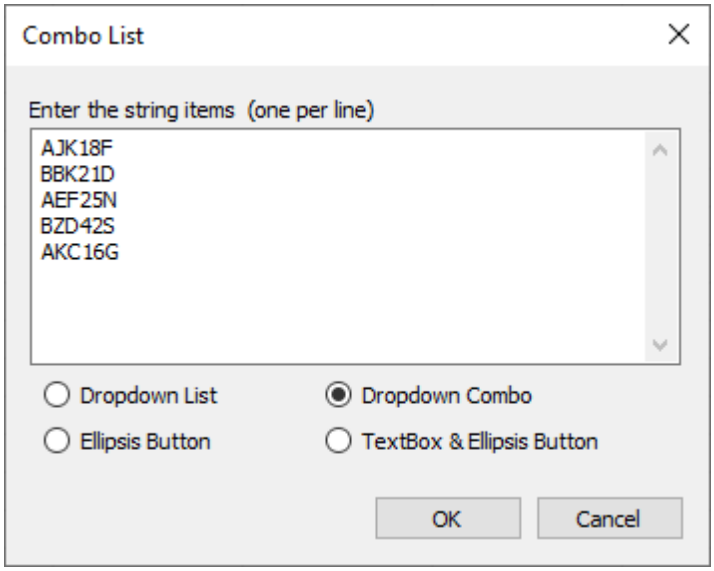
```
// Mark列のComboListプロパティを設定して、セルボタンを表示するようにします
c1flexGrid1.Cols["Mark"].ComboList = "...";

// セルボタンを常に表示できるようにします
c1flexGrid1.Cols["Mark"].ShowButtons = ShowButtonsEnum.Always;

// エンドユーザーがセルボタンをクリックしたときに処理します
c1flexGrid1.CellButtonClick += C1flexGrid1_CellButtonClick;
```

VB.NET

```
' Mark列のComboListプロパティを設定して、セルボタンを表示するようにします
c1flexGrid1.Cols("Mark").ComboList = "..."

' セルボタンを常に表示できるようにします
c1flexGrid1.Cols("Mark").ShowButtons = ShowButtonsEnum.Always

' エンドユーザーがセルボタンをクリックしたときに処理します
c1flexGrid1.CellButtonClick += C1flexGrid1_CellButtonClick
```

ComboList プロパティは、設計時に**[コンボリスト]**ダイアログを使用して設定することもできます。[コンボリスト]ダイアログにアクセスするには、次の手順に従います。

1. エディタを設定する列の**列タスク**メニューを開きます。
2. **[コンボリスト]**オプションに移動して、フィールドの右側にある省略符ボタンをクリックします。
3. [コンボリスト]ダイアログが開きます。このダイアログで、新しい行にオプション値を 1 つずつ指定できます。
4. また、これらの値を**省略符ボタン**または**テキストボックスと省略符ボタン**のどちらで表示するかを選択できます。**[ドロップダウンリスト]**や**[ドロップダウンコンボ]**を作成するオプションもあります。



次の例では、セルボタンをクリックすると、現在の行の背景色と前景色が変わります。同様に、**CellButtonClick** イベントで目的の操作を実行できます。

C#

```
private void C1flexGrid1_CellButtonClick(object sender, RowColEventArgs e)
  {
    // 次のコードは、色を変更して現在の行をマークします
       if (c1flexGrid1.Rows[e.Row].StyleDisplay.BackColor ==
Color.FromName("Window"))
     {
           c1flexGrid1.Rows[e.Row].StyleNew.BackColor = Color.Green;
           c1flexGrid1.Rows[e.Row].StyleNew.ForeColor = Color.White;
       else
           c1flexGrid1.Rows[e.Row].StyleNew.BackColor = Color.FromName("Window");
           c1flexGrid1.Rows[e.Row].StyleNew.ForeColor =
Color.FromName("WindowText");
     }
   }
```

VB.NET

```
Private Sub C1flexGrid1_CellButtonClick(ByVal sender As Object, ByVal e As
RowColEventArgs)
    ' 次のコードは、色を変更して現在の行をマークします
    If c1flexGrid1.Rows(e.Row).StyleDisplay.BackColor Is Color.FromName("Window")
Then
        c1flexGrid1.Rows(e.Row).StyleNew.BackColor = Color.Green
        c1flexGrid1.Rows(e.Row).StyleNew.ForeColor = Color.White

    Else

        c1flexGrid1.Rows(e.Row).StyleNew.BackColor = Color.FromName("Window")
```

```
        c1flexGrid1.Rows(e.Row).StyleNew.ForeColor = Color.FromName("WindowText")
    End If
End Sub
```

セルボタンの画像の変更

デフォルトでは、セルボタンには省略符が表示されます。ただし、CellButtonImage プロパティを使用して、セルボタンの画像を変更できます。

次のコードに示すように、WinForms FlexGrid のセルボタンの画像を設定します。

C#

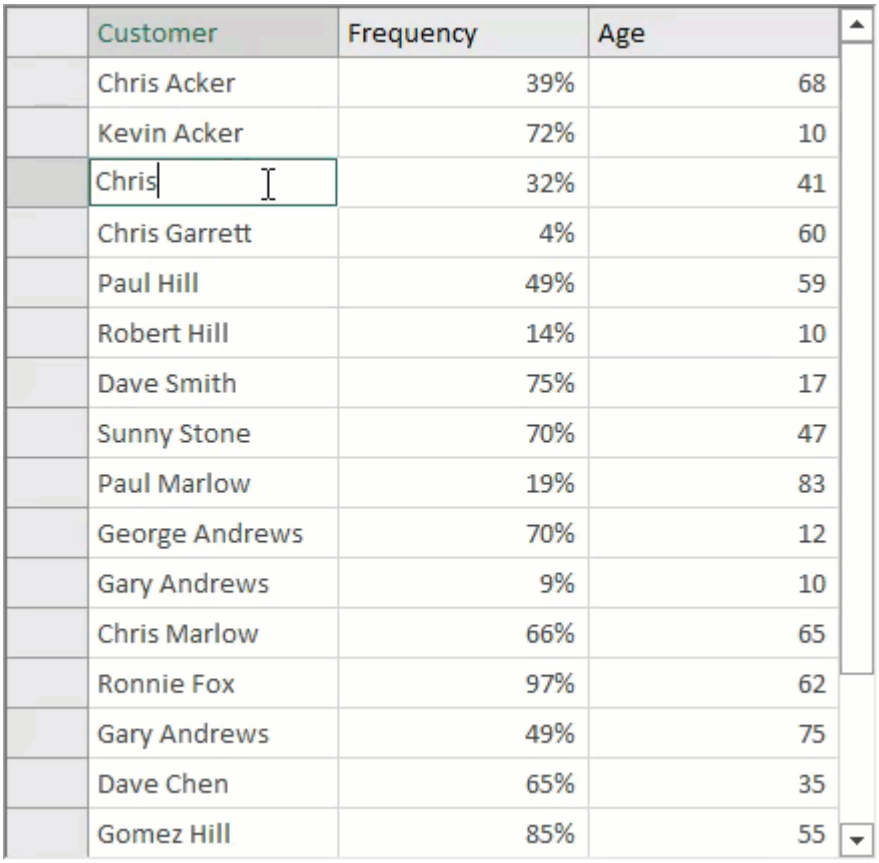
```
// CellButtonImageプロパティを設定して、セルボタンの画像を定義します
Image imgCellBtn = new Bitmap("../../Resources/Images/button.png");
c1flexGrid1.CellButtonImage = imgCellBtn;
```

VB.NET

```
' CellButtonImageプロパティを設定して、セルボタンの画像を定義します
Dim imgCellBtn As Image = New Bitmap("../../Resources/Images/button.png")
c1flexGrid1.CellButtonImage = imgCellBtn
```

検証

データ検証は、ユーザーが列のセルに入力できるデータを制御します。データは、さまざまな方法で検証できます。たとえば、無効なキー入力を制限したり、エラーや警告情報を表示したり、ユーザーから無効な入力値を受け取ったときに元の値に戻すことができます。



	Customer	Frequency	Age
	Chris Acker	39%	68
	Kevin Acker	72%	10
	Chris	32%	41
	Chris Garrett	4%	60
	Paul Hill	49%	59
	Robert Hill	14%	10
	Dave Smith	75%	17
	Sunny Stone	70%	47
	Paul Marlow	19%	83
	George Andrews	70%	12
	Gary Andrews	9%	10
	Chris Marlow	66%	65
	Ronnie Fox	97%	62
	Gary Andrews	49%	75
	Dave Chen	65%	35
	Gomez Hill	85%	55

許容される値の範囲を指定したり、入力文字列の最小の長さを定義するなどの検証を実装するため、FlexGrid には Column クラスの EditorValidation プロパティが用意されています。このプロパティは ValidationRuleCollection クラス型です。このク

ラスは、FlexGrid の列に高度なデータ検証を実装できるように、事前定義されたルールで構成されています。たとえば、次のコードサンプルでは、**EditorValidation** プロパティを使用して、**StringLength** および **Required** の検証ルールが **Customer** 列に適用されています。

次のコードを参照して、WinForms FlexGrid の列に検証ルールを追加する方法を確認してください。

C#

```
private void SetupGridColumns()
{
  var customerNameColumn = _flex.Cols["CustomerName"];
  customerNameColumn.Caption = "Customer";
  customerNameColumn.EditorValidation.Add(new RequiredRule());
  customerNameColumn.EditorValidation.Add(new StringLengthRule()
  {
    MinimumLength = 2
  });
  var customerIDColumn = _flex.Cols["CustomerID"];
  customerIDColumn.Visible = false;
  var frequencyColumn = _flex.Cols["Frequency"];
  frequencyColumn.Format = "0%";
  frequencyColumn.AllowEditing = false;
  var ageColumn = _flex.Cols["Age"];
  ageColumn.EditorValidation.Add(new RequiredRule());
  ageColumn.EditorValidation.Add(new RangeRule()
  {
    Minimum = 10,
    Maximum = 90
  });
}
```

VB.NET

```
Private Sub SetupGridColumns()
    Dim customerNameColumn = _flex.Cols("CustomerName")
    customerNameColumn.Caption = "Customer"
    customerNameColumn.EditorValidation.Add(New RequiredRule())
    customerNameColumn.EditorValidation.Add(New StringLengthRule() With {
        .MinimumLength = 2
    })
    Dim customerIDColumn = _flex.Cols("CustomerID")
    customerIDColumn.Visible = False
    Dim frequencyColumn = _flex.Cols("Frequency")
    frequencyColumn.Format = "0%"
    frequencyColumn.AllowEditing = False
    Dim ageColumn = _flex.Cols("Age")
    ageColumn.EditorValidation.Add(New RequiredRule())
    ageColumn.EditorValidation.Add(New RangeRule() With {
        .Minimum = 10,
        .Maximum = 90
    })
End Sub
```

検証を適用するもう 1 つの方法は、**ValidateEdit** イベントをキャプチャして、**Editor.Text** プロパティの値をチェックすることです。取得した値が無効な場合は、**Cancel** パラメータを true に設定して、ユーザーが有効な値を入力するまで、グリッドを編集モードのままにします。このような検証は、有効値の範囲を設定したり、別のセル値に基づいて現在のセル値を検証するなどのシナリオで使用できます。たとえば、次のサンプルコードでは、通貨列への入力値を検証して、入力された値が 1,000 ～ 10,000 であることを確認しています。

次のコードに示すように、WinForms FlexGrid のセルが編集モードである間、ValidateEdit イベントを使用して有効な値かどう

かをチェックします。

C#

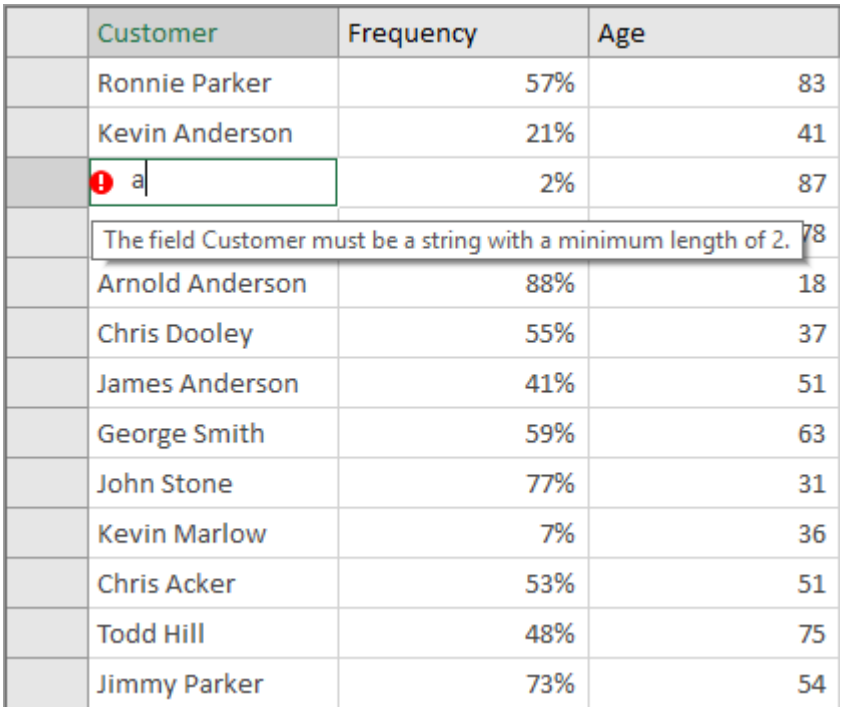
```
private void _flex_ValidateEdit( object sender, ValidateEditEventArgs e)
{

    // 金額を検証します
    if (_flex.Cols[e.Col].DataType == typeof(Decimal))
    {
        try
        {
            Decimal dec = Decimal.Parse(_flex.Editor.Text);
            if ( dec < 1000 || dec > 10000 )
            {
                MessageBox.Show("Value must be between 1,000 and 10,000");
                e.Cancel = true;
            }
        }
        catch
        {
            MessageBox.Show("Value not recognized as a Currency");
            e.Cancel = true;
        }
    }
}
```

VB.NET

```
Private Sub _flex_ValidateEdit(ByVal sender As Object, ByVal e As
ValidateEditEventArgs)

    ' 金額を検証します
    If _flex.Cols(e.Col).DataType Is GetType(Decimal) Then

        Try
            Dim dec As Decimal = Decimal.Parse(_flex.Editor.Text)

            If dec < 1000 OrElse dec > 10000 Then
                MessageBox.Show("Value must be between 1,000 and 10,000")
                e.Cancel = True
            End If

        Catch
            MessageBox.Show("Value not recognized as a Currency")
            e.Cancel = True
        End Try
    End If
End Sub
```

データ注釈

データ注釈とは、クラスなどのオブジェクトに意味のあるメタデータタグを追加することです。これにより、データ検証を実行したり、適切なメッセージをエンドユーザーに表示することで、モデルとビューの間のギャップを簡単に埋めることができます。たとえば、項目をどのように書式設定する必要があるか、どのようなキャプションを付けるべきか、編集可能にするかどうかなどを指定するために、データ注釈を使用できます。



FlexGrid は複数のデータ注釈属性をサポートしており、データクラスのカスタマイズ、ソースからのデータの表示、および検証ルールの設定に使用されます。これらの属性をプロジェクトで使用するには、まず **System.ComponentModel.DataAnnotations** アセンブリへの参照を追加し、次にコードでこれらの属性をデータオブジェクトに追加する必要があります。

> **メモ:** C1FlexGrid は .NET 4 で定義された注釈をサポートします。System.ComponentModel.DataAnnotations はバージョン 4.0.0.0 以上である必要があります。

以下に、FlexGrid コントロールでサポートされている主な注釈属性をリストします。DataAnnotation 属性の完全なリストについては、ここをクリックしてください。

属性名	FlexGrid での機能
Association	エンティティメンバが外部キーリレーションなどのデータリレーションを表すことを指定します。
Display	エンティティ部分クラスの型やメンバに対してローカライズ可能な文字列を指定できる汎用属性を提供します。
DisplayFormat	ASP.NET Dynamic Data によるデータフィールドの表示方法と書式設定方法を指定します。
DisplayColumn	参照されるテーブルで外部キー列として表示される列を指定します。
Editable	データフィールドが編集可能かどうかを示します。
Key	エンティティを一意に識別する 1 つ以上のプロパティを示します。
Validation • RequiredAttribute • StringLengthAttribute • RangeAttribute • RegularExpressionAttribute • MinLengthAttribute • MetaDataAttribute • MaxLengthAttribute	データ注釈の検証属性は、FlexGrid の操作で検証ルールとして使用されます。

属性名	FlexGrid での機能
• EmailAddressAttribute • CompareAttribute • DataTypeAttribute	

次のコード例では、WinForms FlexGrid コントロールでデータ注釈機能がどのように動作するかを示します。

C#

```
// 自動生成されたCustomerName列ヘッダに「Customer」と表示されます
// この列には、少なくとも2つの記号の長さが最小の空でない文字列も必要です

[Display(Name = "Customer")]
[Required]
[StringLength(int.MaxValue, MinimumLength = 2)]

public string CustomerName { get; set; }
// 自動生成されたCustomerID列は非表示になります
[Display(AutoGenerateField = false)]
public int CustomerID { get; set; }
// 自動生成された「Frequency」列には、パーセンテージで書式設定された値が表示されます
// また、編集を許可しません

[DisplayFormat(DataFormatString = "0%")]
[Editable(false)]
public double Frequency { get; set; }
// 自動生成された「Age」列は、事前定義された範囲の値を許可します

[Required]
[Range(10, 90)]
public int Age { get; set; }
// サンプルデータを作成します
public static BindingList<Data> GetSampleData(int cnt)
{
    var list = new BindingList<Data>();
    var rnd = new Random();
    for (int i = 0; i < cnt; i++)
    {
       var item = new Data();
       item.CustomerName = _firstNames[rnd.Next(0, _firstNames.Length)] + " " +
_lastNames[rnd.Next(0, _lastNames.Length)];
       item.CustomerID = i;
       item.Frequency = rnd.NextDouble();
       item.Age = rnd.Next(10, 91);
       list.Add(item);
    }
       return list;
}
```

VB.NET

```
    ' 自動生成されたCustomerName列ヘッダに「Customer」と表示されます
    ' この列には、少なくとも2つの記号の長さが最小の空でない文字列も必要です
    <Display(Name:="Customer")>
    <Required>
    <StringLength(Integer.MaxValue, MinimumLength:=2)>
    Public Property CustomerName As String
    ' 自動生成されたCustomerID列は非表示になります
    <Display(AutoGenerateField:=False)>
    Public Property CustomerID As Integer
```

```
    ' 自動生成された「Frequency」列には、パーセンテージで書式設定された値が表示されます
    ' また、編集を許可しません
    <DisplayFormat(DataFormatString:="0%")>
    <Editable(False)>
    Public Property Frequency As Double
    ' 自動生成された「Age」列は、事前定義された範囲の値を許可します

    <Required>
    <Range(10, 90)>
    Public Property Age As Integer

    ' サンプルデータを作成します
    Public Shared Function GetSampleData(ByVal cnt As Integer) As BindingList(Of
Data)
        Dim list = New BindingList(Of Data)()
        Dim rnd = New Random()

        For i As Integer = 0 To cnt - 1
            Dim item = New Data()
            item.CustomerName = _firstNames(rnd.[Next](0, _firstNames.Length)) & " "
+ _lastNames(rnd.[Next](0, _lastNames.Length))
            item.CustomerID = i
            item.Frequency = rnd.NextDouble()
            item.Age = rnd.[Next](10, 91)
            list.Add(item)
        Next

        Return list
    End Function
```

スパークライン

スパークラインは、周期的な変動などの傾向を示す一連の値を視覚的に表現したり、データ系列の最大値や最小値を強調表示することができる小型のチャートです。スパークラインは、1 つのセルに容易に収まり、また一目でデータパターンを認識できるため、グリッドで特に便利です。

Region	Monthly Sales Trend	Q1 Sales	Q2 Sales	Q3 Sales	Q4 Sales	Quarterly Profit
North		₹ 1,252.00	₹ 2,167.00	₹ 1,494.00	₹ 357.00	
South		₹ 1,509.00	₹ 1,480.00	₹ 1,458.00	₹ 1,785.00	
East		₹ 1,659.00	₹ 1,247.00	₹ 2,151.00	₹ 2,460.00	
West		₹ 1,042.00	₹ 2,079.00	₹ 1,364.00	₹ 1,389.00	

FlexGrid for WinForms では、Column クラスの ShowSparkline プロパティに **true** を設定することで、列にスパークラインを表示できます。また、Sparkline クラスの **SparklineType** プロパティを使用して、スパークラインのタイプを設定できます。FlexGrid は、以下の 3 種類のスパークラインをサポートします。

- **折れ線**:折れ線グラフと同様に、折れ線スパークラインは、折れ線を利用してある期間の値の変化を示します。
- **縦棒**:縦棒グラフと同様に、縦棒スパークラインは、縦棒を使用して、複数のカテゴリにわたるパターンを示します。
- **勝敗**:縦棒スパークラインと同様に、勝敗スパークラインは縦棒を使用して値を表します。ただし、大きさは表さず、通常は 2 値データを示すために使用されます。

スパークラインにデータマーカーを表示するには、**ShowMarkers** プロパティに **true** を設定します。また、**Sparkline** クラスには、最初、最後、最高、最低、および負のデータポイントをそれぞれ異なる書式で強調表示するためのプロパティもあります。**Styles** プロパティを使用して、スパークラインのスタイルを設定することもできます。

以下のコードは、WinForms FlexGrid の列にスパークラインを追加する方法を示します。

C#

```
// WinLossSparklineを表示するようにプロパティを設定します
c1FlexGrid1.Cols[2].ShowSparkline = true;
c1FlexGrid1.Cols[2].Sparkline.SparklineType = SparklineType.WinLoss;
c1FlexGrid1.Cols[2].Sparkline.Styles.SeriesColor = Color.Green;
c1FlexGrid1.Cols[2].Sparkline.ShowNegative = true;
```

VB.NET

```
' WinLossSparklineを表示するようにプロパティを設定します
c1FlexGrid1.Cols(2).ShowSparkline = True
c1FlexGrid1.Cols(2).Sparkline.SparklineType = SparklineType.WinLoss
c1FlexGrid1.Cols(2).Sparkline.Styles.SeriesColor = Color.Green
c1FlexGrid1.Cols(2).Sparkline.ShowNegative = True
```

ヘッダーとフッター

列ヘッダーとは、グリッドの上部に固定される行で、キャプション文字列、ソートグリフなどが表示されます。

FlexGrid では、デフォルトで、インデックスがゼロの上端の行が列ヘッダーに割り当てられます。ただし、他の行も固定されるように指定して、ヘッダーを広げることができます。複数の行を固定に設定するには、RowCollection クラスの **Fixed** プロパティに 1 より大きい整数を設定する必要があります。

	Company	Sales				Purchases			
		Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
	[2,1]	[2,2]	[2,3]	[2,4]	[2,5]	[2,6]	[2,7]	[2,8]	[2,9]
	[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	[3,6]	[3,7]	[3,8]	[3,9]
	[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	[4,6]	[4,7]	[4,8]	[4,9]
	[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	[5,6]	[5,7]	[5,8]	[5,9]
	[6,1]	[6,2]	[6,3]	[6,4]	[6,5]	[6,6]	[6,7]	[6,8]	[6,9]
	[7,1]	[7,2]	[7,3]	[7,4]	[7,5]	[7,6]	[7,7]	[7,8]	[7,9]
	[8,1]	[8,2]	[8,3]	[8,4]	[8,5]	[8,6]	[8,7]	[8,8]	[8,9]
	[9,1]	[9,2]	[9,3]	[9,4]	[9,5]	[9,6]	[9,7]	[9,8]	[9,9]
	[10,1]	[10,2]	[10,3]	[10,4]	[10,5]	[10,6]	[10,7]	[10,8]	[10,9]

ヘッダーテキストの設定

連結モードの場合、FlexGrid は、データソースからフィールド名を読み取って、列ヘッダーテキストとしてレンダリングします。ただし、Row クラスの **Caption** プロパティを明示的に設定して、データソースのフィールド名文字列を上書きできます。非連結グリッドの場合も、**Caption** プロパティでヘッダーテキストを指定します。このプロパティは、インデックスがゼロのデフォルトのヘッダー行のセルに値を設定します。他の固定行のセルに値を設定するには、FlexGrid の通常の値の割り当て方法を使用する必要があります。セル値の設定方法の詳細については、「セルに値を設定」を参照してください。

WinForms FlexGrid で以下のコードを使用して、ヘッダー行を指定し、ヘッダーテキストを設定します。

C#

```
// 2つの行を列ヘッダー行として設定します
c1FlexGrid1.Rows.Fixed = 2;

// 最初の列のヘッダーとサブヘッダーを設定します
c1FlexGrid1.Cols[1].Caption = "Column Header 1";
c1FlexGrid1[1, 1] = "Column Sub-header 1";
```

VB.NET

```
' 2つの行を列ヘッダー行として設定します
c1FlexGrid1.Rows.Fixed = 2

' 最初の列のヘッダとサブヘッダーを設定します
c1FlexGrid1.Cols(1).Caption = "Column Header 1"
c1FlexGrid1(1, 1) = "Column Sub-header 1"
```

列ヘッダーの結合

FlexGrid には、特定の行（この場合はヘッダー行）のセルが結合可能かを指定する **Row** オブジェクトの **AllowMerging** プロパティがあります。ヘッダー行の結合を許可したら、C1FlexGrid クラスの AllowMerging プロパティまたは AllowMergingFixed プロパティに **FixedOnly** を設定します。FlexGrid コントロールにはこの 2 つのプロパティがあるため、結合に関連する複数のロジックをより柔軟に実装できます。セルの結合の詳細については、「結合」を参照してください。

以下のコードを使用して、WinForms FlexGrid の列ヘッダーを結合します。

C#

```
// ヘッダ行でのマージを許可します
c1FlexGrid1.Rows[0].AllowMerging = true;

// グリッドのAllowMergingまたはAllowMergingFixedプロパティを設定して、固定行のみを結合します
// c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly;
c1FlexGrid1.AllowMergingFixed = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly;
```

VB.NET

```
' ヘッダ行でのマージを許可します
c1FlexGrid1.Rows(0).AllowMerging = True

' グリッドのAllowMergingまたはAllowMergingFixedプロパティを設定して、固定行のみを結合します
' c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly
c1FlexGrid1.AllowMergingFixed = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly
```

列ヘッダーテキストの折り返し

列ヘッダーのテキストを折り返すには、CellStyleCollection クラスのCellStyle 項目 "**Fixed**" にアクセスし、その WordWrap プロパティに **true** を設定します。折り返したテキストを正しく表示するには、行の高さを調整するか、AutoSizeRow() メソッドを呼び出してテキストの長さに基づいて自動的に行のサイズを変更する必要があります。

以下のコードを使用して、WinForms FlexGrid の列ヘッダーテキストを折り返します。

C#

```
//列のキャプションを設定します
c1FlexGrid1.Cols[3].Caption = "Large Text for Column Header Text Wrapping";

//行の高さを設定します
c1FlexGrid1.Rows[0].Height = 50;

//固定の行と列のワードラップを設定します
c1FlexGrid1.Styles["Fixed"].WordWrap = true;
```

VB.NET

```
'列のキャプションを設定します
c1FlexGrid1.Cols(3).Caption = "Large Text for Column Header Text Wrapping"

'行の高さを設定します
c1FlexGrid1.Rows(0).Height = 50

'固定の行と列のワードラップを設定します
c1FlexGrid1.Styles("Fixed").WordWrap = True
```

列ヘッダーのスタイル設定

列ヘッダーのスタイルを設定するには、**CellStyleCollection** クラスの **CellStyle** 項目 "**Fixed**" にアクセスし、**Font**、**ForeColor**、**TextEffect** などのさまざまなスタイル設定に関連するプロパティを設定します。

以下のコードを使用して、WinForms FlexGrid の列ヘッダーをカスタマイズします。

C#

```
//ヘッダーテキストのフォントを設定します
c1FlexGrid1.Styles["Fixed"].Font = new Font("Tahoma", 10, FontStyle.Bold);
```

```
//ヘッダーテキストの前色を設定します
c1FlexGrid1.Styles["Fixed"].ForeColor = Color.Aqua;

//ヘッダーテキストの背景色を設定します
c1FlexGrid1.Styles["Fixed"].BackColor = Color.Blue;

//ヘッダーテキストにテキスト効果を適用します
c1FlexGrid1.Styles["Fixed"].TextEffect = C1.Win.C1FlexGrid.TextEffectEnum.Raised;
```

VB.NET

```
' ヘッダテキストのフォントを設定します
c1FlexGrid1.Styles("Fixed").Font = New Font("Tahoma", 10, FontStyle.Bold)

' ヘッダテキストの前色を設定します
c1FlexGrid1.Styles("Fixed").ForeColor = Color.Aqua

' ヘッダーテキストの背景色を設定します
c1FlexGrid1.Styles("Fixed").BackColor = Color.Blue

' ヘッダテキストにテキスト効果を適用します
c1FlexGrid1.Styles("Fixed").TextEffect = C1.Win.C1FlexGrid.TextEffectEnum.Raised
```

列フッターの設定

列フッターはグリッドの最後の行で、列全体に関する追加的な情報が表示されます。列フッターの最も一般的な使用方法は、列データの概要を表示することです。

FlexGrid では、**C1FlexGrid** クラスの **Footers** プロパティを使用して列フッターを作成できます。FlexGrid では、Footers クラスの **Fixed** プロパティを使用して、行と一緒にフッターをスクロールするか、それともグリッドの下部に固定するかを選択します。このクラスには **Descriptions** プロパティもあり、ここから FooterDescription コレクションにアクセスして、**キャプション**などの追加情報を設定します。さまざまな集計関数を通して列フッターに集計結果を表示するには、**FooterDescription** クラスの **Aggregates** プロパティを使用して、AggregateDefinition コレクションにアクセスする必要があります。集計関数は、AggregateEnum 列挙の値を受け取る Aggregate プロパティを使用して指定できます。

以下のコードは、WinForms FlexGrid に列フッターを追加する方法を示します。

C#

```
 // 列にフッタを追加します
c1FlexGrid1.Footers.Descriptions.Add(new C1.Win.C1FlexGrid.FooterDescription() { Caption = "Total"
});

// フッタに集計を適用します
c1FlexGrid1.Footers.Descriptions[0].Aggregates.Add(new C1.Win.C1FlexGrid.AggregateDefinition()
{ Column = 4, Aggregate = C1.Win.C1FlexGrid.AggregateEnum.Sum });
```

VB.NET

```
' 列にフッタを追加します
c1FlexGrid1.Footers.Descriptions.Add(New C1.Win.C1FlexGrid.FooterDescription() With {
            .Caption = "Total"
        })

' フッタに集計を適用します
c1FlexGrid1.Footers.Descriptions(0).Aggregates.Add(New C1.Win.C1FlexGrid.AggregateDefinition() With
{
            .Column = 4,
            .Aggregate = C1.Win.C1FlexGrid.AggregateEnum.Sum
        })
```

サイズ変更

列幅の設定

FlexGrid には、グリッドの列幅を設定するために ColumnCollection クラスの DefaultSize プロパティがあります。Column クラスの **Width** プロパティを設定することで、特定の列の幅を指定することもできます。**Width** プロパティのデフォルト値は -1 です。これは、列が **DefaultSize** プロパティによって指定される幅になることを意味しています。設計時に **C1FlexGrid 列エ**

ディタで Width プロパティを使用することもできます。列エディタの詳細については、「エディタ」を参照してください。

以下のコードを使用して、WinForms FlexGrid の列のデフォルトの幅を設定します。

C#

```
// すべての列のデフォルト幅を設定します
c1FlexGrid1.Cols.DefaultSize = 110;

// 最初の列の幅を設定します
c1FlexGrid1.Cols[1].Width = 30;
```

VB.NET

```
' すべての列のデフォルト幅を設定します
c1FlexGrid1.Cols.DefaultSize = 110

' 最初の列の幅を設定します
c1FlexGrid1.Cols(1).Width = 30
```

列幅の自動調整

テキスト長に基づいて列の幅を調整するために、FlexGrid には **AutoSizeCol()** メソッドと **AutoSizeCols()** メソッドがあります。**AutoSizeCol()** メソッドは、指定された列の幅を自動的に調整します。一方、**AutoSizeCols()** メソッドはセル範囲に対して使用されます。

以下のコードは、WinForms FlexGrid でテキスト長に基づいて列の幅を自動的に調整する方法を示しています。

C#

```
// テキストの長さに応じて最初の列の幅を自動調整します
c1FlexGrid1.AutoSizeCol(1);

// 列の幅を1番目から4番目まで自動調整します
c1FlexGrid1.AutoSizeCols(1, 4, 2);

// すべての列の幅を自動調整します
// c1FlexGrid1.AutoSizeCols();
```

VB.NET

```
' テキストの長さに応じて最初の列の幅を自動調整します
c1FlexGrid1.AutoSizeCol(1)

' 列の幅を1番目から4番目まで自動調整します
c1FlexGrid1.AutoSizeCols(1, 4, 2)

' すべての列の幅を自動調整します
' c1FlexGrid1.AutoSizeCols()
```

最小/最大列幅の設定

FlexGrid では、ColumnCollection の MinSize プロパティと MaxSize プロパティを使用して、列の幅の範囲を設定できます。この機能は、**AllowResizing** プロパティに true を設定した場合、または **AutoSizeCol** または **AutoSizeCols** メソッドを使用する場合などに特に便利です。

以下のコードを使用して、WinForms FlexGrid で列の幅の範囲を指定します。

C#

```
// 列コレクションの最小幅を設定します
c1FlexGrid1.Cols.MinSize = 20;

// 列コレクションの最大幅を設定します
c1FlexGrid1.Cols.MaxSize = 60;
```

VB.NET

```
' 列コレクションの最小幅を設定します
c1FlexGrid1.Cols.MinSize = 20

' 列コレクションの最大幅を設定します
c1FlexGrid1.Cols.MaxSize = 60
```

スターサイズ

スターサイズは、グリッドのサイズが変更されてもレイアウトが変わらないように、グリッドの列幅の比率を維持したまま列をサイズ変更して、利用可能なスペースの全体を使用することです。たとえば、スターサイズが "*"、"3*"、"2*"、"*"、および "*" で指定された 5 列のグリッドがあるとします。この場合、先頭、4 番目、および 5 番目の列は常に同じ幅になります。また、2 番目の列は先頭の列の 3 倍、3 番目の列は 2 倍の幅になります。

	Department	Task	TaskGroup	Target	Done
	Marketing	Market research meeting	Finato	12/15/2013	☑
	R&D	Meet with Spanish research team	Jaleo	12/4/2013	☑
	Operations	Plans for new plant complete	Atlantis	12/6/2013	☑
	Production	Production feasilibity report	Finato	12/18/2013	☐
	Marketing	Brainstorming session	Titan/2	12/7/2013	☑
	R&D	Feasibility session	Titan/2	12/9/2013	☑
	Legal	Draft plan to exit GTS service agreement	Geronimo	12/10/2013	☐
	Accounting	Final budget review	Finato	12/15/2013	☑
	Operations	Send out bid forms	Atlantis	12/18/2013	☐
	R&D	Fallback planning	Jaleo	12/19/2013	☐
	Production	Production planning	Finato	12/20/2013	☐

列のスターサイズを指定するために、FlexGrid には、Column クラスの **StarWidth** プロパティがあります。幅が狭くなりすぎたり、広くなりすぎることを防ぐには、**MinWidth** プロパティと **MaxWidth** プロパティを設定できます。これらのプロパティは、**C1FlexGrid 列エディタ**からも使用できます。列エディタの詳細については、「エディタ」を参照してください。

以下のコードを使用して、WinForms FlexGrid の列でスターサイズ、すなわち比率を維持したサイズ変更を行います。

C#

```
//列のスターサイズを設定します
c1FlexGrid1.Cols[1].StarWidth = "*";
c1FlexGrid1.Cols[2].StarWidth = "3*";
c1FlexGrid1.Cols[3].StarWidth = "2*";
c1FlexGrid1.Cols[4].StarWidth = "*";
c1FlexGrid1.Cols[5].StarWidth = "*";

// 列が狭くなりすぎないように、MinWidthプロパティを設定します
c1FlexGrid1.Cols[1].MinWidth = 50;
```

VB.NET

```
' 列のスターサイズを設定します
```

```
c1FlexGrid1.Cols(1).StarWidth = "*"
c1FlexGrid1.Cols(2).StarWidth = "3*"
c1FlexGrid1.Cols(3).StarWidth = "2*"
c1FlexGrid1.Cols(4).StarWidth = "*"
c1FlexGrid1.Cols(5).StarWidth = "*"

' 列が狭くなりすぎないように、MinWidthプロパティを設定します
c1FlexGrid1.Cols(1).MinWidth = 50
```

列バンド

FlexGrid allows you to organize columns into logical groups, known as **Column Bands**, which can be used to create Banded grid view, where the data is represented in a tabular form with columns arranged into Bands. The **Banded View** makes it easier to represent huge amount of data in the grid. For example, suppose you want to display the order and shipping details of products in a grid. In this case, you can create a banded view to organize data in a hierarchical structure and manage it easily.

	− Order		− Shipping			
	OrderID	OrderDate	ShippedDate	ShipVia	ShipName	ShipAddress
	10248	04-07-2006	16-07-2006	3	Vins et alcools Chev	59 rue de l'Abbay
	10249	05-07-2006	10-07-2006	1	Toms Spezialitäte	Luisenstr. 48
	10250	08-07-2006	12-07-2006	2	Hanari Carnes	Rua do Paço, 67
	10251	08-07-2006	15-07-2006	1	Victuailles en stock	2, rue du Commer
	10252	09-07-2006	11-07-2006	2	Suprêmes délices	Boulevard Tirou, 2
	10253	10-07-2006	16-07-2006	2	Hanari Carnes	Rua do Paço, 67
	10254	11-07-2006	23-07-2006	2	Chop-suey Chinese	Hauptstr. 31
	10255	12-07-2006	15-07-2006	3	Richter Supermarkt	Starenweg 5
	10256	15-07-2006	17-07-2006	2	Wellington Importa	Rua do Mercado,
	10257	16-07-2006	22-07-2006	3	HILARION-Abastos	Carrera 22 con Av
	10258	17-07-2006	23-07-2006	1	Ernst Handel	Kirchgasse 6
	10259	18-07-2006	25-07-2006	3	Centro comercial M	Sierras de Granac
	10260	19-07-2006	29-07-2006	1	Ottilies Käselade	Mehrheimerstr. 36
	10261	19-07-2006	30-07-2006	2	Que Delícia	Rua da Panificado
	10262	22-07-2006	25-07-2006	3	Rattlesnake Canyon	2817 Milton Dr.

The **Column Bands** feature of the FlexGrid uses the **C1FlexGridBandedView** class of **C1.Win.FlexGrid** namespace of **C1.Win.FlexGrid.BandedView** assembly. Users can add the **C1.Win.FlexGrid.BandedView** NuGet package using the **NuGet Package Manager**. This will add the NuGet packages under the **Dependencies** section in the **Solution Explorer**. On switching to the Design View, you will find the **FlexGridBandedView** component in the ToolBox, which when added to the designer gets added in the ComponentTray.

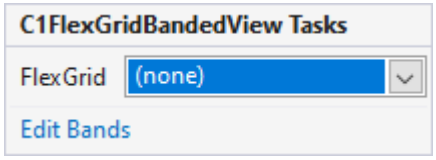
The right pane displays a toolbar of band operation buttons towards the top such as Add Band, Add child Band, Add parent Band and Remove Band.

Configure FlexGridBandedView via Smart Tag Panel

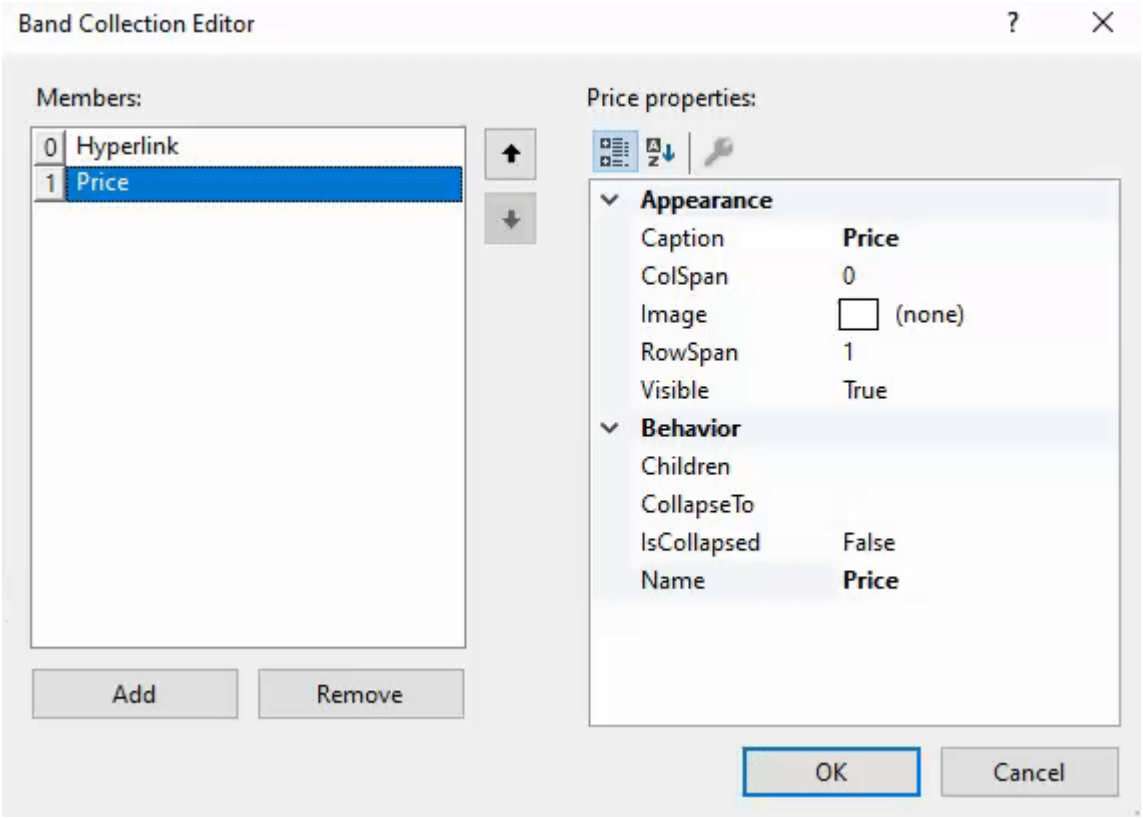
This section helps you configure banded view in FlexGrid control using **C1FlexGridBandedView Tasks** smart tag panel in .NET and .NET Framework Editions.

.NET

Clicking the FlexGridBandedView smart tag icon (▶) opens the **C1FlexGridBandedView Tasks** smart tag panel as shown in the following image.



The **C1FlexGridBandedView Tasks** smart tag panel lets you bind the FlexGridBandedView with a FlexGrid control, and edit the band settings using the **Band Collection Editor**. It also lets you create multi-level bands by adding multiple child bands to a parent band.

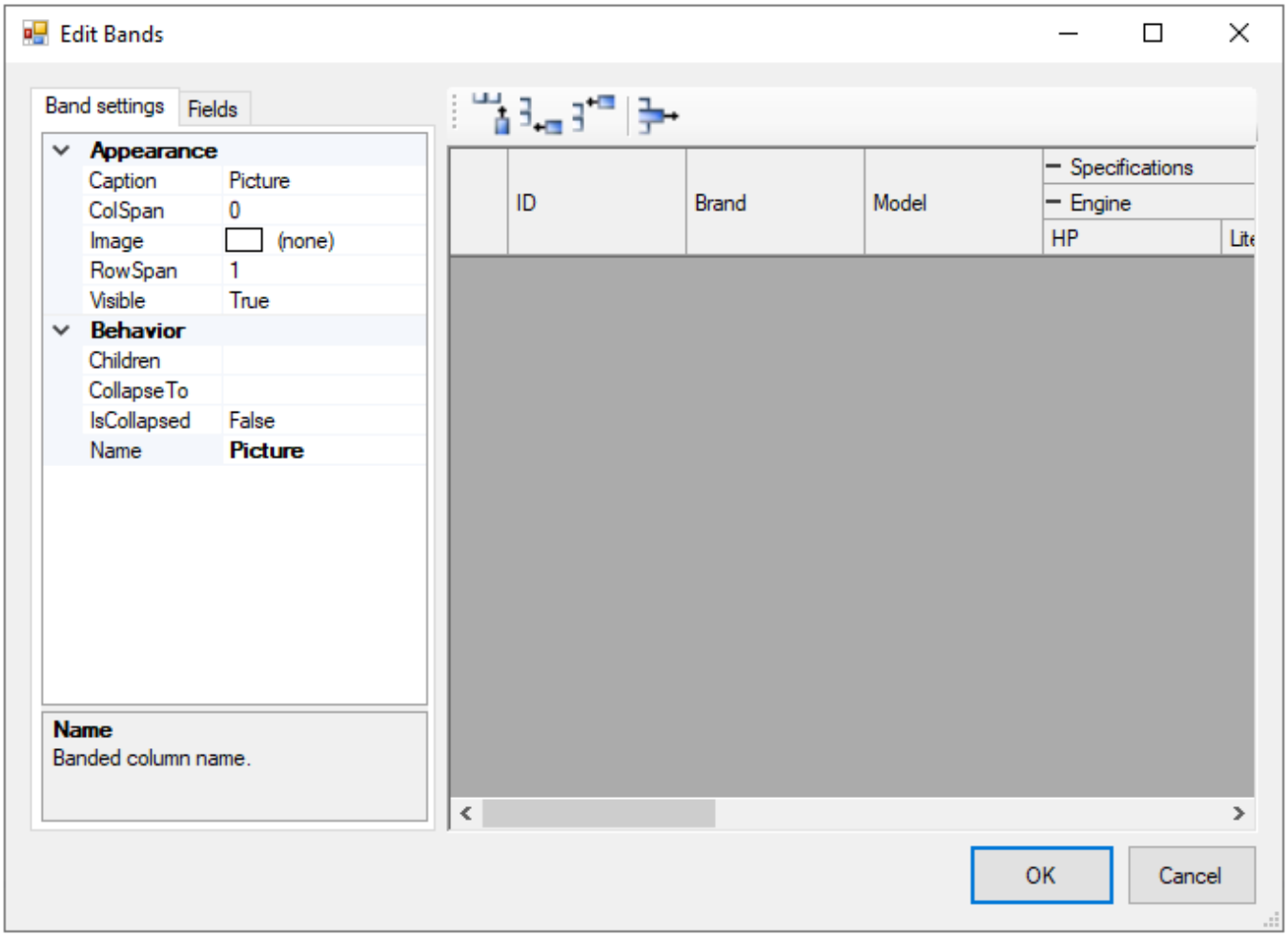


In the **Band Collection Editor** window, the right pane displays properties like Caption, ColSpan, RowSpan, Image, Visibility etc. for each band and the left pane displays the list of bands along with the **Add** and **Remove** buttons which can be used to add and remove bands, respectively. All changes post the addition, deletion or customization of bands from the **Band Collection Editor** are reflected in the band fields.

.NET Framework

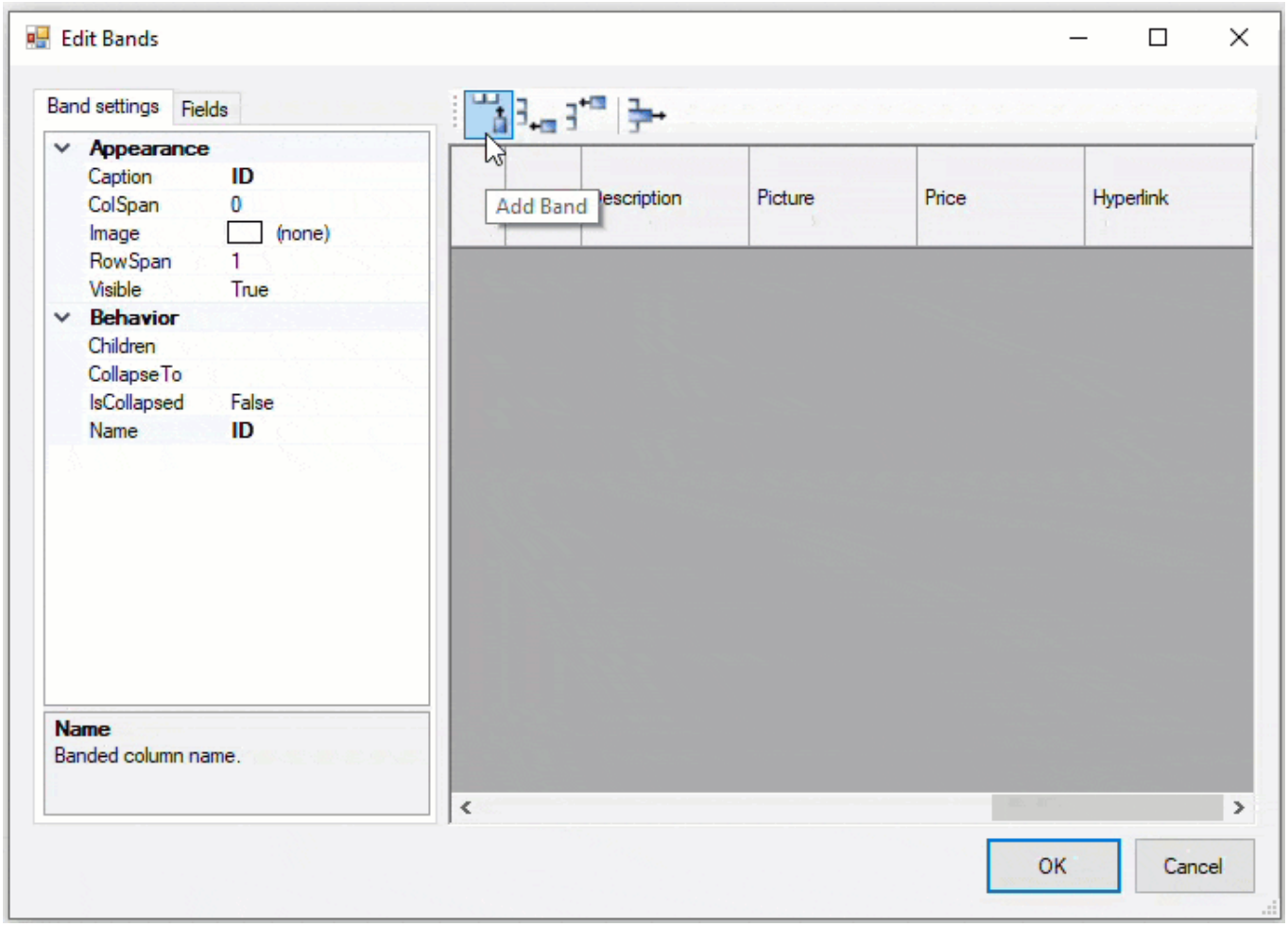
Clicking on the FlexGridBandedView smart tag, opens the FlexGridBandedView Tasks smart tag panel.

The Tasks smart tag panel lets you bind the FlexGridBandedView with a FlexGrid control, and edit the band settings and fields using the Bands Editor.



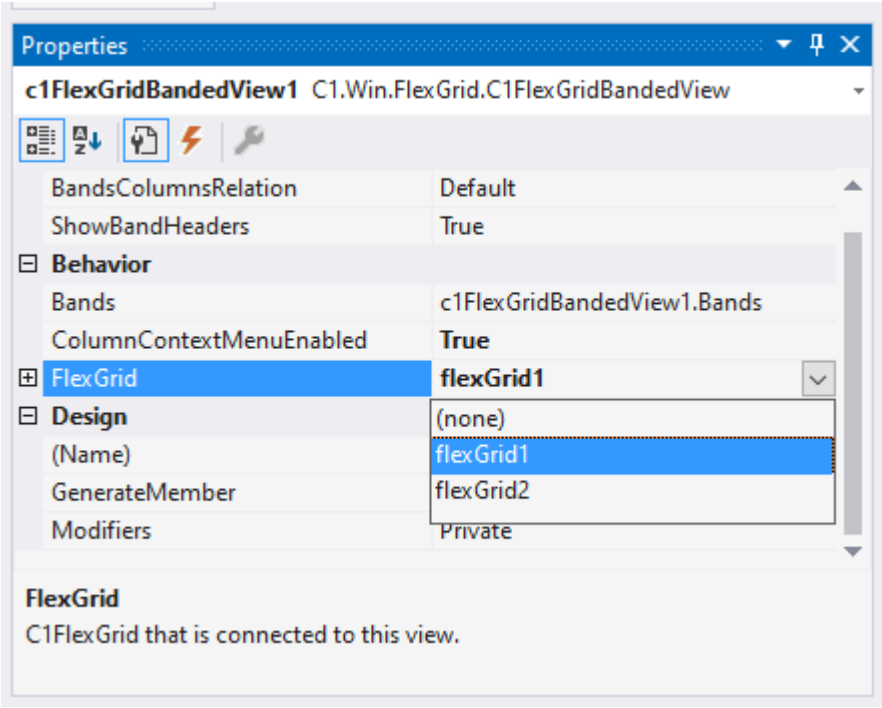
The Bands Editor has two panes. The left pane provides two tabs, Band settings and Fields. The Band settings tab lets you customize properties like Caption, ColSpan, RowSpan, Image, Visibility etc. for each band. The Fields tab lets you check or uncheck the band names. The right pane displays a toolbar of band operation buttons towards the top such as Add Band, Add child Band, Add parent Band and Remove Band. Just below the toolbar in the right pane is the band fields. All changes post the addition, deletion or customization of bands are reflected in the band fields.

Observe the GIF below to understand the working of the Bands Editor.



Configure FlexGridBandedView via Properties Window

If you observe the properties of the **FlexGridBandedView** component in the **Properties** window, you will find the **FlexGrid** property, which allows you to assign a preferred FlexGrid control to the component.



Configure FlexGridBandedView via Code

You can also set the **FlexGrid** property in the code behind as given below:

C#

```
// FlexGridコントロールを割り当てます
c1FlexGridBandedView1.FlexGrid = flexGrid1;
```

The banded view is formed by arranging columns into bands wherein a collection of two or more individual columns are placed under a common header. In FlexGrid, the concept of Banded view is implemented using MergedRanges and FixedRows. You can create a Banded view by assigning columns to the respective Bands. This section contains information about banded columns and the operations that can be performed on them.

Create Column Bands

You can create bands by using the **Bands** property of **C1FlexGridBandedView** class and **Add()** method of **BandCollection** class. You can also use the **AddRange( )** method of the **BandCollection** class to add a range of columns.

The below code snippet shows how you can add a column to the band.

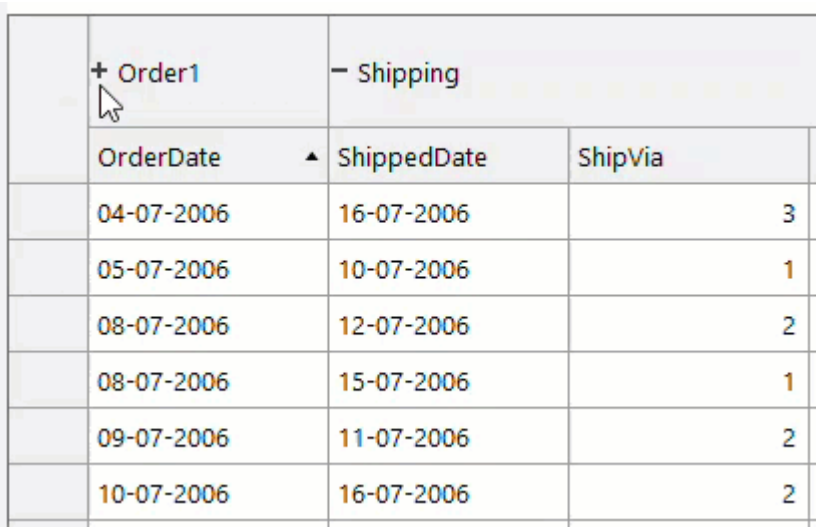
C#

```
var bands = c1FlexGridBandedView1.Bands;
var band1 = bands.Add("Order");
```

Header Band

Header Band represents the header for logically grouped columns. You can assign any number of Banded columns to a Header Band. You can set the following properties of Header Band.

- **Set Header Text**: To provide a name to the Header Band, use the **Caption** property of the **Band** class.
- **Set RowSpan**: To set the minimum number of rows to be occupied by the Band, use **RowSpan** property of the **Band** class.
- **Set ColSpan**: To set the maximum number of columns to be occupied by the Band, use **ColSpan** property of the **Band** class.
- **Set Visibility**: To make the Header Band visible, set the **Visible** property of the **Band** class to true. Similarly, you can also hide a Header Band by setting Visible property to false.
- **Expand/Collapse**: To collapse a Band horizontally, you can use the **CollapseTo** property of the Band class. If the CollapseTo column is unavailable (i.e., hidden or moved), then the column gets collapsed to the first available column in the Band. You can also use context menu for collapsing the Band at the design-time. On the other hand, you can set the Visible property of the Band class to false to collapse a band vertically.



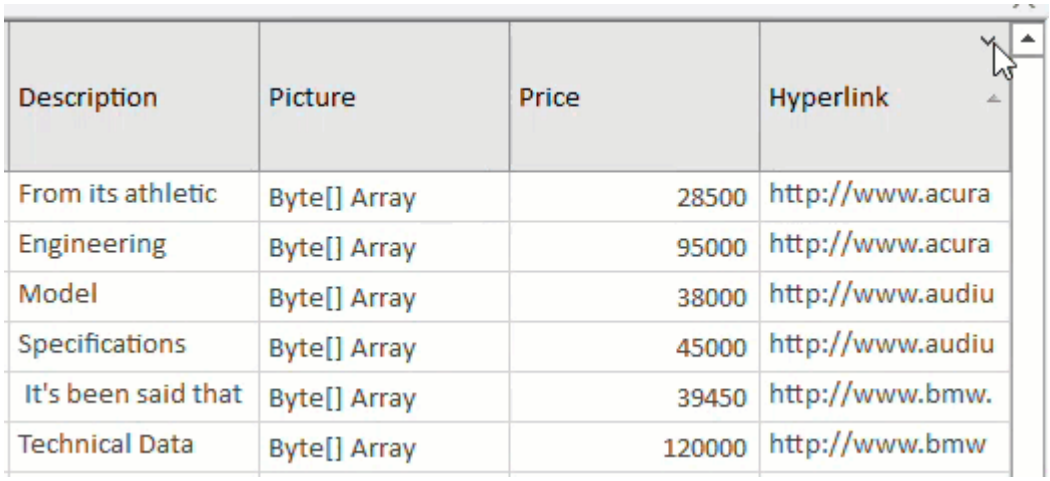
	+ Order1	− Shipping	
	OrderDate ▲	ShippedDate	ShipVia
	04-07-2006	16-07-2006	3
	05-07-2006	10-07-2006	1
	08-07-2006	12-07-2006	2
	08-07-2006	15-07-2006	1
	09-07-2006	11-07-2006	2
	10-07-2006	16-07-2006	2

The code snippet below depicts the use of these properties:

C#

```
var bands = c1FlexGridBandedView1.Bands;
var band1 = bands.Add("Order");
band1.Children.Add("OrderID");
band1.Children.Add("OrderDate");
band1.CollapseTo = "OrderDate";
band1.Caption = "Order1";
band1.RowSpan = 2;
band1.ColSpan = 2;
band1.Visible = true;
```

Further, at runtime, users can vertically collapse the header using the chevron button.



Description	Picture	Price	Hyperlink
From its athletic	Byte[] Array	28500	http://www.acura
Engineering	Byte[] Array	95000	http://www.acura
Model	Byte[] Array	38000	http://www.audiu
Specifications	Byte[] Array	45000	http://www.audiu
It's been said that	Byte[] Array	39450	http://www.bmw.
Technical Data	Byte[] Array	120000	http://www.bmw

Create Multi-Level Bands

FlexGrid allows you to create multi-level bands by stacking multiple bands or by adding multiple child bands to a parent band. In a multi-level band, the span of rows increases with the increasing level of Bands, making the Band with the highest level expands by several rows (provided RowSpan is zero). The lowest band in multi-level band is the band column, which corresponds to the grid column and does not contain any child bands. You can create Multi-level bands by adding child bands to the parent band's Children Collection. To do so, you can use the **Children** property of the **Band** class.

The below code snippet shows how you can create multi-level bands.

C#

```
band5.Caption = "Engine";
band6.Caption = "HP";
band6.Name = "HP";
band7.Caption = "Liter";
band7.Name = "Liter";
band8.Caption = "Cyl";
band8.Name = "Cyl";
band5.Children.Add(band6);
band5.Children.Add(band7);
band5.Children.Add(band8);
```

Customize Banded View

In a banded view, the columns outside the bands are placed after the bands. However, this presentation can be changed using the **BandsColumnsRelation** property of the **Band** class. The BandsColumnsRelation property uses **BandsColumnsRelation** enumeration to change the positioning of bands and columns in a Banded layout by setting one of the following values:

- **Default**: Displays the columns after the bands. It is the default value.
- **Bands**: Displays only the bands.
- **BandsBeforeColumns**: Displays the columns after the bands.
- **ColumnBeforeBands**: Displays columns before the bands.

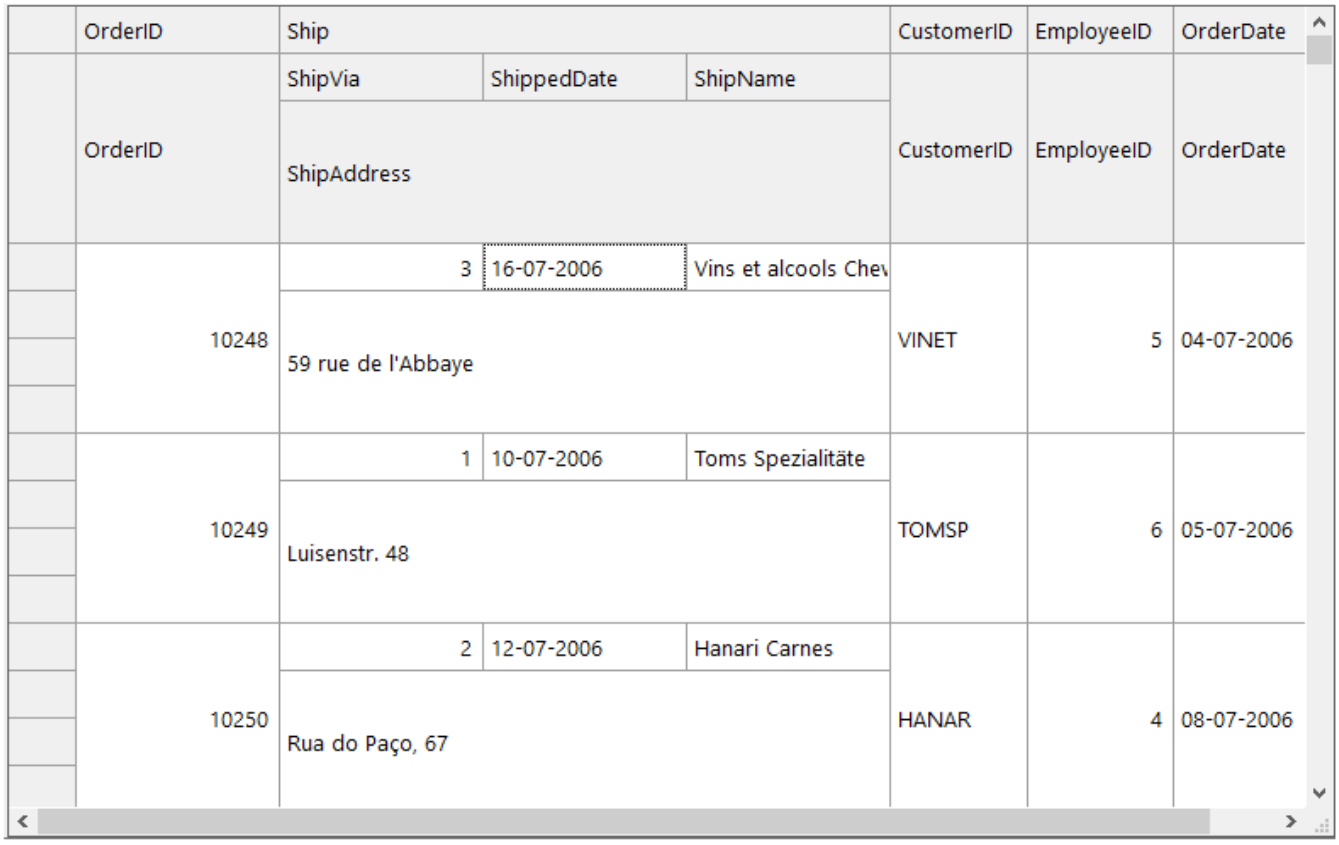
The below code snippet shows how you can change the column layout in the Banded Column.

C#

```
c1FlexGridBandedView1.BandsColumnsRelation = BandsColumnsRelation.Bands;
```

Advanced Column Bands

The **FlexGridBandedView** takes up the advanced banded view depending upon the Band settings (that is, the RowSpan and ColSpan properties). In **Advanced Banded View,** bands are displayed in multiple rows. It supports complex cell layout, which provides a better view. In this view, the grid caption is divided into two sections: Header Band and Record Bands. Here, the Header Band represents the header for logically grouped columns, while the Record Bands represent logical groups of columns, which are linked to specific columns. In an Advanced banded view, the bands can occupy multiple rows and columns. You can set the minimum number of rows occupied by a band in this view using the **RowSpan** property of the **Band** class, and set the maximum number of columns occupied by the band using the **ColSpan** property of the **Band** class.



	OrderID	Ship			CustomerID	EmployeeID	OrderDate
	OrderID	ShipVia	ShippedDate	ShipName	CustomerID	EmployeeID	OrderDate
		ShipAddress					
	10248	3	16-07-2006	Vins et alcools Chev	VINET	5	04-07-2006
		59 rue de l'Abbaye					
	10249	1	10-07-2006	Toms Spezialitäte	TOMSP	6	05-07-2006
		Luisenstr. 48					
	10250	2	12-07-2006	Hanari Carnes	HANAR	4	08-07-2006
		Rua do Paço, 67					

The following code snippet shows setting an advanced banded view.

C#

```
var bands = c1FlexGridBandedView1.Bands;
bands.Add("OrderID");
var bMain = bands.Add("Ship");
bMain.ColSpan = 3;
bMain.Children.Add("ShipVia");
bMain.Children.Add("ShippedDate");
bMain.Children.Add("ShipName");
var bDescription = bMain.Children.Add("ShipAddress");
bDescription.ColSpan = 3;
bDescription.RowSpan = 3;
bands.Add("CustomerID");
bands.Add("EmployeeID");
bands.Add("OrderDate");
```

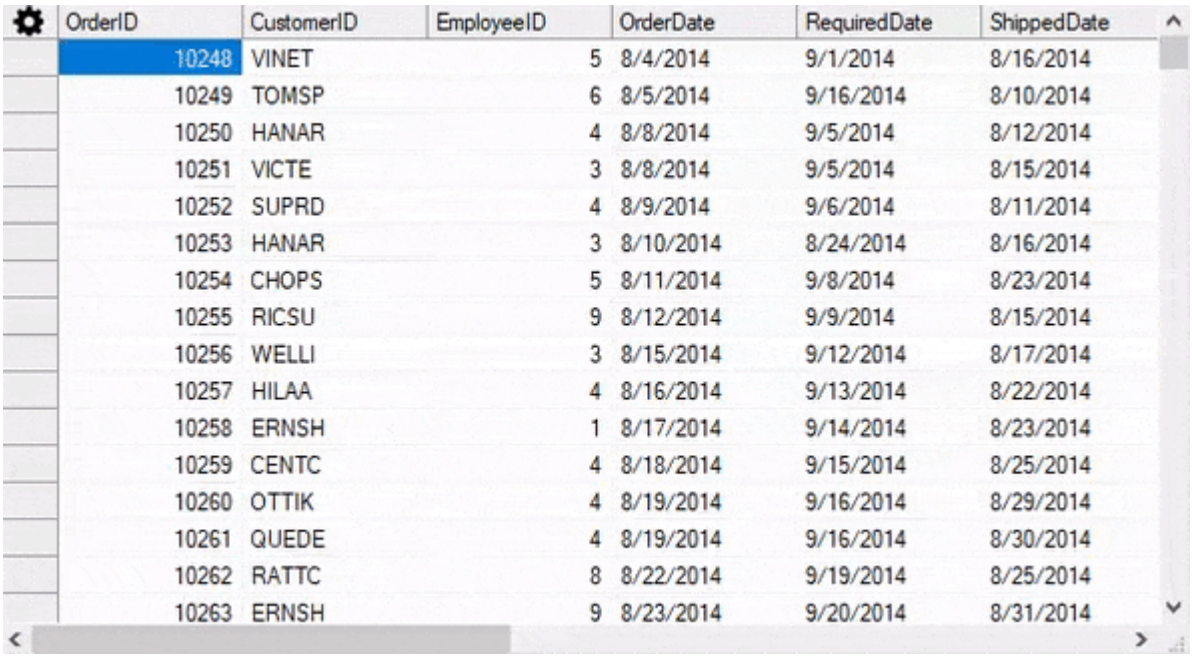
列ピッカー

FlexGrid provides a built-in column picker that can simply be enabled by setting **ShowToolButton** property of the **ColumnPickerInfo** class to **true**. Enabling column picker allows end-users to add or remove a visible column through the gear icon () on the top left corner of the grid. The following code demonstrates the use of **ShowToolButton** property to enable column picker in FlexGrid:

C#

```
c1FlexGrid1.ColumnPickerInfo.ShowToolButton = true;
```

The following GIF shows how you can remove some visible columns using the Column Picker.



OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate
10248	VINET	5	8/4/2014	9/1/2014	8/16/2014
10249	TOMSP	6	8/5/2014	9/16/2014	8/10/2014
10250	HANAR	4	8/8/2014	9/5/2014	8/12/2014
10251	VICTE	3	8/8/2014	9/5/2014	8/15/2014
10252	SUPRD	4	8/9/2014	9/6/2014	8/11/2014
10253	HANAR	3	8/10/2014	8/24/2014	8/16/2014
10254	CHOPS	5	8/11/2014	9/8/2014	8/23/2014
10255	RICSU	9	8/12/2014	9/9/2014	8/15/2014
10256	WELLI	3	8/15/2014	9/12/2014	8/17/2014
10257	HILAA	4	8/16/2014	9/13/2014	8/22/2014
10258	ERNSH	1	8/17/2014	9/14/2014	8/23/2014
10259	CENTC	4	8/18/2014	9/15/2014	8/25/2014
10260	OTTIK	4	8/19/2014	9/16/2014	8/29/2014
10261	QUEDE	4	8/19/2014	9/16/2014	8/30/2014
10262	RATTC	8	8/22/2014	9/19/2014	8/25/2014
10263	ERNSH	9	8/23/2014	9/20/2014	8/31/2014

Alternatively, column picker can be enabled through column context menus. You just need to select the **Open Column Picker** option from the column context menu to open the Column Picker UI. However, you need to set **ShowColumnMenuItem** property to **true** to display the **Open Column Picker** option in column context menu as shown in the following code.

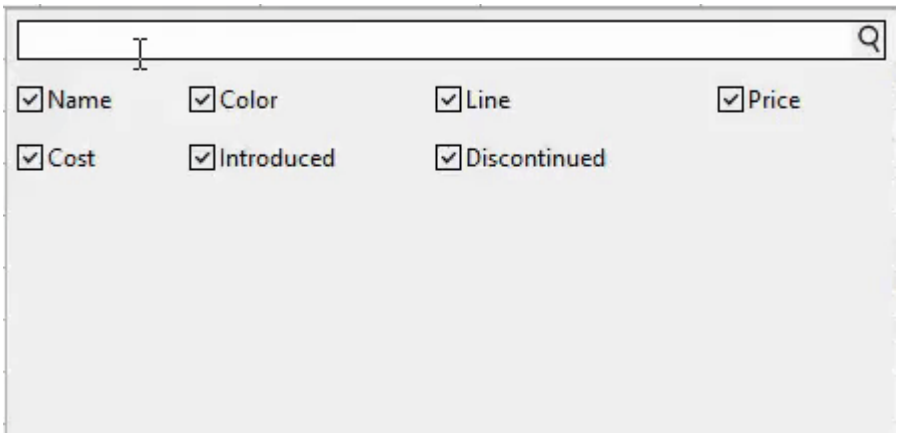
C#

```
c1FlexGrid1.ColumnContextMenuEnabled = true;
c1FlexGrid1.ColumnPickerInfo.ShowColumnMenuItem = true;
```

Note: You need to enable the column context menu first in order to display **Open Column Picker** option in context menu.
For more information on column context menu, see **Enable Context menu** section of the User Interaction topic.

Search

With Column Picker, you can easily search for a column from the displayed column list. You can do so by entering the column name in the text field, and clicking the **Search** button as shown in the following GIF.



To enable column picker search, you need to use the **SearchMode** property which sets the type of column picker search through the **ColumnPickerSearchMode** enumeration. The **ColumnPickerSearchMode** enumeration provides

the following options for the search type:

Enumeration Value	Description
None	Disables the search panel.
Highlight	Highlights the search results.

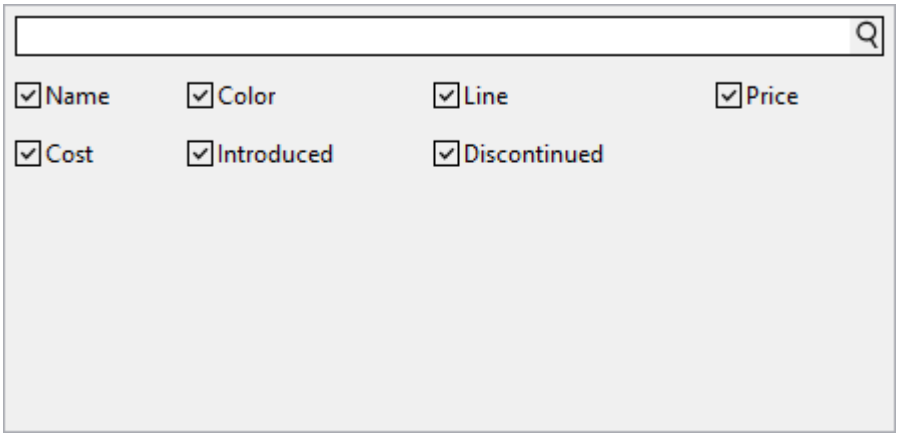
The following code snippet shows how to enable search in the Column Picker.

C#

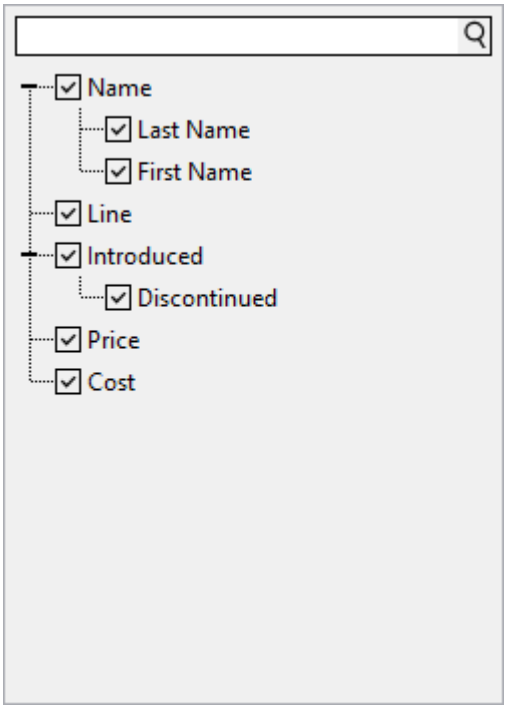
```
c1FlexGrid1.ColumnPickerInfo.SearchMode =
C1.Win.FlexGrid.ColumnPickerSearchMode.Highlight;
```

Views

Column Picker provides different in-built views, such as list view and tree view based on the hierarchy of column fields in FlexGrid. By default, Column Picker displays column list in the form of list view as shown in the following image.



Alternatively, the column list is displayed in the form of tree view when hierarchical data is displayed in the columns, i.e., columns are arranged into bands as shown in the following image.



Note: This view only appears when FlexGrid is integrated with **C1FlexGridBandedView** component.

行

グリッドコントロールは、行と列のコレクションです。一般に、列にはそれぞれ特定の種類の情報が格納されます。一方、行は、それぞれ特定の項目に関するさまざまな種類の情報を記録するために使用されます。

FlexGrid では、行のコレクションは **RowCollection** クラスで表されます。これにアクセスするには、**C1FlexGrid** クラスの **Rows** プロパティを使用します。このセクションでは、行で実行できるさまざまな操作について説明します。

トピック	スナップショット	コンテンツ
基本的な操作		基本的な行の操作方法について説明します。 • 行数の設定 • 行の追加 • 行の削除 • 行の挿入 • データ型の設定 • 固定行の設定 • フリーズ行の設定
ユーザー操作		実行時にエンドユーザーが実行できる操作について説明します。 • 追加の許可 • 削除の許可 • ドラッグの許可 • フリーズの許可 • サイズ変更の許可
ヘッダー		行ヘッダーの設定およびその他の関連する操作を実行する方法について説明します。 • ヘッダーテキストの設定 • 行ヘッダーの結合 • 行ヘッダーテキストの折り返し • 行ヘッダーのスタイル設定
行詳細		行詳細機能を使用して追加情報を表示する方法について説明します。 • フォーム内編集 • 階層ビュー • カスタム行詳細
サイズ変更		行のサイズを変更するさまざまな方法について説明します。 • 行の高さの設定 • 行の高さの自動調整 • 行の最小/最大高さの設定

基本的な操作

このトピックでは、行で実行可能なさまざまな基本操作について説明します。

行数の設定

グリッドがデータソースに連結されている場合、行の数はデータソースにあるレコードの数によって決まります。ただし、非連結モードの場合は、RowCollection クラスの **Count** プロパティに整数値を設定して、グリッドに表示する行の数を指定できます。

以下のコードを使用して、WinForms FlexGrid の行数を設定します。

C#

```
// 行数を設定します
c1FlexGrid1.Rows.Count = 15;
```

VB.NET

```
' 行数を設定します
```

```
c1FlexGrid1.Rows.Count = 15
```

行の追加

FlexGrid には、実行時に新しい行を追加するさまざまな方法があります。新しいレコードを追加するには、**Add** メソッド（**RowCollection** クラス）または **AddItem** メソッド（C1FlexGrid クラス）のいずれかを使用します。非連結モードでは、**Count** プロパティの値をインクリメントして、新しい行を追加することもできます。これらの方法はすべて、グリッドの末尾に向かって行を追加します。特定の位置に行を挿入する方法については、「**行の挿入**」を参照してください。また、**Count** プロパティを使用して連結グリッドに新しい行を追加すると、例外が生成されることに注意してください。

以下のコードに示すアプローチのいずれかを使用して、WinForms FlexGrid に行を追加します。

C#

```
// 方法 1:
// Use the RowCollection.Add method to add row
 C1.Win.C1FlexGrid.Row r;
 r = c1FlexGrid1.Rows.Add();

// データを設定します
 r[1] = "New Row 2";

 // 方法 2:
 // 行を追加してデータを設定するには、PASSWORDメソッドを使用します
  c1FlexGrid1.AddItem("" + "\t" + "New Row 1");

// 方法 3:
// RowCollection.Countプロパティを使用して行を追加します
 c1FlexGrid1.Rows.Count += 1;

// データを設定します
 c1FlexGrid1[c1FlexGrid1.Rows.Count - 1, 1] = "New Row 3";
```

VB.NET

```
' 方法 1:
' Use the RowCollection.Add method to add row
Dim r As C1.Win.C1FlexGrid.Row
r = c1FlexGrid1.Rows.Add()

' データを設定します
r(1) = "New Row 2"

' 方法 2:
' 行を追加してデータを設定するには、PASSWORDメソッドを使用します
c1FlexGrid1.AddItem("" & vbTab & "New Row 1")

' 方法 3:
' RowCollection.Countプロパティを使用して行を追加します
c1FlexGrid1.Rows.Count += 1

' データを設定します
c1FlexGrid1(c1FlexGrid1.Rows.Count - 1, 1) = "New Row 3"
```

行の削除

グリッドから特定の行を削除するには、**RowCollection** クラスの **Remove** メソッドを使用して、削除する行をそのパラメータとして指定します。RowCollection クラスには、行の範囲を 1 回の呼び出しで削除できる **RemoveRange** メソッドもあります。同

様に、**C1FlexGrid** クラスの **RemoveItem** メソッドを使用して、特定の行を削除することもできます。非連結グリッドでは、**Count** プロパティの値を変更することで、行の数を減らすことができます。

次のコードは、さまざまな方法で WinForms FlexGrid から行を削除します。

C#

```
// 方法 1:
// RowCollection.Removeメソッドを使用して2番目の行を削除します
c1FlexGrid1.Rows.Remove(2);

// 方法 2:
// RemoveItemメソッドを使用して2番目の行を削除します
c1FlexGrid1.RemoveItem(2);

// 方法 3:
// RemoveRangeメソッドを使用して、2番目から始まる3行を削除します
c1FlexGrid1.Rows.RemoveRange(2, 3);

// 方法 4:
// RowCollection.Countプロパティを使用して最後の行を削除します
c1FlexGrid1.Rows.Count -= 1;
```

VB.NET

```
' 方法 1:
' RowCollection.Removeメソッドを使用して2番目の行を削除します
c1FlexGrid1.Rows.Remove(2)

' 方法 2:
' RemoveItemメソッドを使用して2番目の行を削除します
c1FlexGrid1.RemoveItem(2)

' 方法 3:
' RemoveRangeメソッドを使用して、2番目から始まる3行を削除します
c1FlexGrid1.Rows.RemoveRange(2, 3)

' 方法 4:
' RowCollection.Countプロパティを使用して最後の行を削除します
c1FlexGrid1.Rows.Count -= 1
```

行の挿入

FlexGrid の特定の位置に行を挿入するには、**RowCollection** クラスの Insert メソッドを使用し、このメソッドで行を挿入する位置を指定します。また、グリッドに複数の行を挿入するには、**InsertRange** メソッドを使用します。

以下のコードは、WinForms FlexGrid の特定の位置に行を挿入する方法を示しています。

C#

```
C1.Win.C1FlexGrid.Row r;

// 方法 1:
// Insertメソッドを使用して、2番目の位置に行を挿入します
r = c1FlexGrid1.Rows.Insert(2);
r[1] = "Inserted row";

// 方法 2:
// InsertRangeメソッドを使用して、2番目の位置に3つの行を追加します
// c1FlexGrid1.Rows.InsertRange(2, 3);
```

VB.NET

```
Dim r As C1.Win.C1FlexGrid.Row

 ' 方法 1:
 ' Insertメソッドを使用して、2番目の位置に行を挿入します
 r = c1FlexGrid1.Rows.Insert(2)
 r(1) = "Inserted row"

 ' 方法 2:
 ' InsertRangeメソッドを使用して、2番目の位置に3つの行を追加します
 ' c1FlexGrid1.Rows.InsertRange(2, 3)
```

データ型の設定

連結 FlexGrid の場合、各連結列のデータ型は、データに応じてデータソースから自動的に取得されます。ただし、非連結モードの場合は、Row クラスまたは Column クラスの DataType プロパティを指定して、行または列のデータ型をそれぞれ設定できます。行と列の両方にデータ型が設定されている場合は、列の設定の方が行の設定より優先されます。

WinForms FlexGrid の行のデータ型を設定するには、次のコードを使用します。

C#

```
C1.Win.C1FlexGrid.Row r;
r = c1FlexGrid1.Rows.Add();
r.DataType = typeof(int);
```

VB.NET

```
Dim r As C1.Win.C1FlexGrid.Row
r = c1FlexGrid1.Rows.Add()
r.DataType = GetType(Integer)
```

固定行の設定

固定行とは、編集不可のセルを含む行です。この行は、ユーザーがグリッドを下にスクロールしても、常にグリッドの最上部に表示されます。FlexGrid で固定行を設定するには、**RowCollection** クラスの **Fixed** プロパティを使用します。このプロパティは、固定する行の数を指定する整数値を受け取ります。

WinForms FlexGrid で固定行を設定するには、次のコードを使用します。

C#

```
// 3行を固定行として設定します
c1FlexGrid1.Rows.Fixed = 3;
```

VB.NET

```
' 3行を固定行として設定します
c1FlexGrid1.Rows.Fixed = 3
```

フリーズ行の設定

フリーズ行は、固定行と同様にスクロールできません。ただし、ユーザーはこの行を編集できます。FlexGrid でフリーズ行を設定するには、RowCollection クラスにある **Frozen** プロパティを使用します。

以下のコードを使用して、WinForms FlexGrid のフリーズ行を設定します。

C#

```
// 最初の2行をフリーズ行として設定します
c1FlexGrid1.Rows.Frozen = 2;
```

VB.NET

```
' 最初の2行をフリーズ行として設定します
c1FlexGrid1.Rows.Frozen = 2
```

ユーザー操作

このトピックでは、エンドユーザーに FlexGrid の行の操作を許可する方法について説明します。

追加の許可

FlexGrid は、デフォルトでは、グリッドへの新しい行の追加をエンドユーザーに許可しません。実行時に行を追加するためのオプションを提供するには、C1FlexGrid クラスの **AllowAddNew** プロパティを **true** に設定します。FlexGrid では、**C1FlexGrid タスク**メニューに**[行の追加可能]**という設計時オプションも用意されており、これを使用してエンドユーザーに新しい行の追加を許可することもできます。タスクメニューの詳細については、「タスクメニュー」を参照してください。さらに、**NewRowWatermark** プロパティを使用して、新しい行テンプレートに表示するウォーターマークテキストを設定することもできます。

次のコードを使用すると、WinForms FlexGrid で実行時の行追加をユーザーに許可できます。

C#

```
// ユーザーが行を追加できるようにします
 c1FlexGrid1.AllowAddNew = true;

// テンプレートの新しい行にウォーターマークを設定します
 c1FlexGrid1.NewRowWatermark = "Add new row";
```

VB.NET

```
 ' ユーザーが行を追加できるようにします
 c1FlexGrid1.AllowAddNew = True

 ' テンプレートの新しい行にウォーターマークを設定します
 c1FlexGrid1.NewRowWatermark = "Add new row"
```

削除の許可

FlexGrid は、デフォルトでは、グリッドからの行の削除をエンドユーザーに許可しません。ただし、アプリケーションで必要な場合は、**AllowDelete** プロパティを **true** に設定することで、エンドユーザーが**[Del]**キーを使用して選択した行を削除できるようにすることができます。また、**C1FlexGrid タスク**メニューで**[行の削除可能]**チェックボックスをオンにして、行の削除を可能にすることもできます。

次のコードは、実行時に WinForms FlexGrid からの行の削除をユーザーに許可する方法を示しています。

C#

```
// ユーザーが行を削除できるようにします
c1FlexGrid1.AllowDelete = true;
```

VB.NET

```
' ユーザーが行を削除できるようにします
c1FlexGrid1.AllowDelete = True
```

ドラッグを許可

FlexGrid は、デフォルトでは、ドラッグによる行の並べ替えをユーザーに許可しません。ただし、非連結グリッドでは、FlexGrid.AllowDragging プロパティと Row.AllowDragging プロパティを使用してこの動作を変更できます。グリッド行のドラッグを可能にするには、**FlexGrid.AllowDragging** プロパティを **Rows** または **Both** に設定します。このプロパティは、AllowDraggingEnum に含まれる値を受け取ります。また、**Row.AllowDragging** プロパティを **false** に設定することで、特定の行のドラッグを禁止することもできます。

メモ: 連結モードでは行の移動は許可されないため、連結グリッドで行をドラッグすると、例外が生成されます。 連結モードでのドラッグの実装については、ブログ記事「Drag and Drop Rows in C1Flexgrid（C1Flexgrid での行のドラッグアンドドロップ）」を参照してください。

次のコードを使用して、非連結 WinForms FlexGrid でユーザーが実行時に行をドラッグすることを許可できます。

C#

```
// グリッド全体ですべての行のドラッグを許可します
 c1FlexGrid1.AllowDragging = C1.Win.C1FlexGrid.AllowDraggingEnum.Rows;

// 特定の行のドラッグを無効にします
 c1FlexGrid1.Rows[3].AllowDragging = false;
```

VB.NET

```
 ' グリッド全体ですべての行のドラッグを許可します
 c1FlexGrid1.AllowDragging = C1.Win.C1FlexGrid.AllowDraggingEnum.Rows

 ' 特定の行のドラッグを無効にします
 c1FlexGrid1.Rows(3).AllowDragging = False
```

フリーズの許可

実行時にエンドユーザーが行をフリーズできるようにするには、C1FlexGrid クラスの AllowFreezing プロパティを使用します。これは、AllowFreezingEnum に含まれる値を受け取ります。このプロパティが **Rows** または **Both** に設定されている場合は、マウスポインタをヘッダー行の端に置くと鍵のアイコンが表示されます。エンドユーザーは、鍵のアイコンをクリックしてドラッグすることで行をフリーズできます。

次のコードを使用すると、実行時にユーザーが WinForms FlexGrid の行をフリーズすることを許可できます。

C#

```
// 実行時に行のフリーズを許可します
c1FlexGrid1.AllowFreezing = C1.Win.C1FlexGrid.AllowFreezingEnum.Rows;
```

VB.NET

```
' 実行時に行のフリーズを許可します
c1FlexGrid1.AllowFreezing = C1.Win.C1FlexGrid.AllowFreezingEnum.Rows
```

サイズ変更を許可

FlexGrid は、デフォルトでは、行をサイズ変更するオプションを提供していません。この動作を変更するには、**C1FlexGrid** クラスの AllowResizing プロパティを使用します。このプロパティは AllowResizingEnum 列挙の値を受け取り、エンドユーザーが列、行、またはその両方をサイズ変更できるようにします。この列挙では、行、列、またはその両方のサイズを一様に変更することもできます。つまり、1 つの列または行のサイズを変更すると、残りの列と行のサイズも自動的に変更されます。また、FlexGrid にはブール型の Row.AllowResizing プロパティもあります。これを使用して、特定の行のサイズ変更を有効/無効にできます。

次のコードは、実行時に WinForms FlexGrid の行のサイズ変更をユーザーに許可する方法を示しています。

C#

```
// ユーザーが行のサイズを変更できるようにします
c1FlexGrid1.AllowResizing = C1.Win.C1FlexGrid.AllowResizingEnum.Rows;

// ユーザーによる2行目のサイズ変更を停止します
c1FlexGrid1.Rows[2].AllowResizing = false;
```

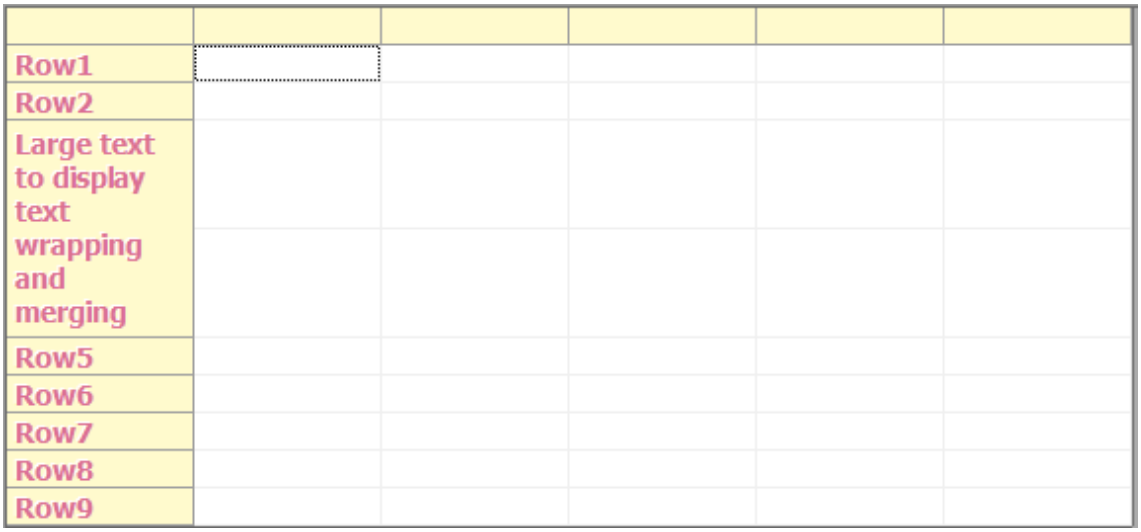
VB.NET

```
' ユーザーが行のサイズを変更できるようにします。
c1FlexGrid1.AllowResizing = C1.Win.C1FlexGrid.AllowResizingEnum.Rows

' ユーザーによる2行目のサイズ変更を停止します
c1FlexGrid1.Rows(2).AllowResizing = False
```

ヘッダー

行ヘッダーは、グリッドの左側にある 1 つまたは複数の固定行です。ここには、キャプション文字列が含まれている場合も、そうでない場合もあります。

FlexGrid では、デフォルトで、インデックスがゼロの左端の列が行ヘッダーに割り当てられます。ただし、他の列も固定されるように指定して、ヘッダーを広げることができます。 複数の列を固定に設定するには、ColumnCollection クラスの **Fixed** プロパティに 1 より大きい整数を設定する必要があります。



ヘッダーテキストの設定

行ヘッダーテキストを設定するには、Row クラスの **Caption** プロパティを設定します。このプロパティは、インデックスがゼロのデフォルトのヘッダー列のセルに値を設定します。他の固定列のセルに値を設定するには、FlexGrid の通常の値の割り当て方法を使用する必要があります。セル値の設定方法の詳細については、「セルに値を設定」を参照してください。

WinForms FlexGrid で以下のコードを使用して、ヘッダー列を指定し、ヘッダーテキストを設定します。

C#

```
// 最初の行のヘッダーを設定します
c1FlexGrid1.Rows[1].Caption = "Row 1";

// すべての行にヘッダーを設定します
for (int i = c1FlexGrid1.Rows.Fixed; i < c1FlexGrid1.Rows.Count; i++)
  {
     c1FlexGrid1[i, 0] = "Row" + i.ToString();
  }

// 列幅を設定します
c1FlexGrid1.Cols[0].Width = 85;
```

VB.NET

```
' 最初の行のヘッダーを設定します
c1FlexGrid1.Rows(1).Caption = "Row 1"

' すべての行にヘッダーを設定します
For i As Integer = c1FlexGrid1.Rows.Fixed To c1FlexGrid1.Rows.Count - 1
    c1FlexGrid1(i, 0) = "Row" & i.ToString()
Next

' 列幅を設定します
c1FlexGrid1.Cols(0).Width = 85
```

行ヘッダーの結合

FlexGrid には、特定の列（この場合はヘッダー列）のセルが結合可能かを指定する Column クラスの **AllowMerging** プロパティがあります。ヘッダー列の結合を許可したら、C1FlexGrid クラスの AllowMerging プロパティまたは AllowMergingFixed プロパティに **FixedOnly** を設定します。FlexGrid コントロールにはこの 2 つのプロパティがあるため、結合に関連する複数のロジックをより柔軟に実装できます。セルの結合の詳細については、「結合」を参照してください。

以下のコードを使用して、WinForms FlexGrid の行ヘッダーを結合します。

C#

```
// ヘッダー列での結合を許可します
c1FlexGrid1.Cols[0].AllowMerging = true;

// グリッドのAllowMergingまたはAllowMergingFixedプロパティを設定して、固定の行/列のみをマージします
// c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly;
c1FlexGrid1.AllowMergingFixed = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly;
```

VB.NET

```
' ヘッダー列での結合を許可します
c1FlexGrid1.Cols(0).AllowMerging = True

' グリッドのAllowMergingまたはAllowMergingFixedプロパティを設定して、固定の行/列のみをマージします
' c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly
c1FlexGrid1.AllowMergingFixed = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly
```

行ヘッダーテキストの折り返し

行ヘッダーのテキストを折り返すには、CellStyleCollection クラスのCellStyle 項目 "**Fixed**" にアクセスし、その WordWrap プロパティに **true** を設定します。折り返されたテキストが適切に表示されるようにするには、FlexGrid の行の高さを調整する必要があります。または、AutoSizeRow() メソッドを呼び出して、テキストの長さに応じて行が自動的にサイズ変更されるようにします。

以下のコードを使用して、WinForms FlexGrid の行ヘッダーテキストを折り返します。

C#

```
// 行のキャプションを設定します
c1FlexGrid1.Rows[3].Caption = "Large text to display text wrapping and merging";
c1FlexGrid1.Rows[4].Caption = "Large text to display text wrapping and merging";

// 行の高さを設定します
c1FlexGrid1.Rows[3].Height = 35;
c1FlexGrid1.Rows[4].Height = 35;

// 固定の行と列の折り返しを設定します
c1FlexGrid1.Styles["Fixed"].WordWrap = true;
```

VB.NET

```
' 行のキャプションを設定します
c1FlexGrid1.Rows(3).Caption = "Large text to display text wrapping and merging"
c1FlexGrid1.Rows(4).Caption = "Large text to display text wrapping and merging"

' 行の高さを設定します
c1FlexGrid1.Rows(3).Height = 35
c1FlexGrid1.Rows(4).Height = 35

' 固定の行と列の折り返しを設定します
c1FlexGrid1.Styles("Fixed").WordWrap = True
```

行ヘッダーのスタイル設定

行ヘッダーのスタイルを設定するには、**CellStyleCollection** クラスの **CellStyle** 項目 "**Fixed**" にアクセスし、**Font**、**ForeColor**、**TextEffect** などのさまざまなスタイル設定に関連するプロパティを設定します。

以下のコードを使用して、WinForms FlexGrid の行ヘッダーをカスタマイズします。

C#

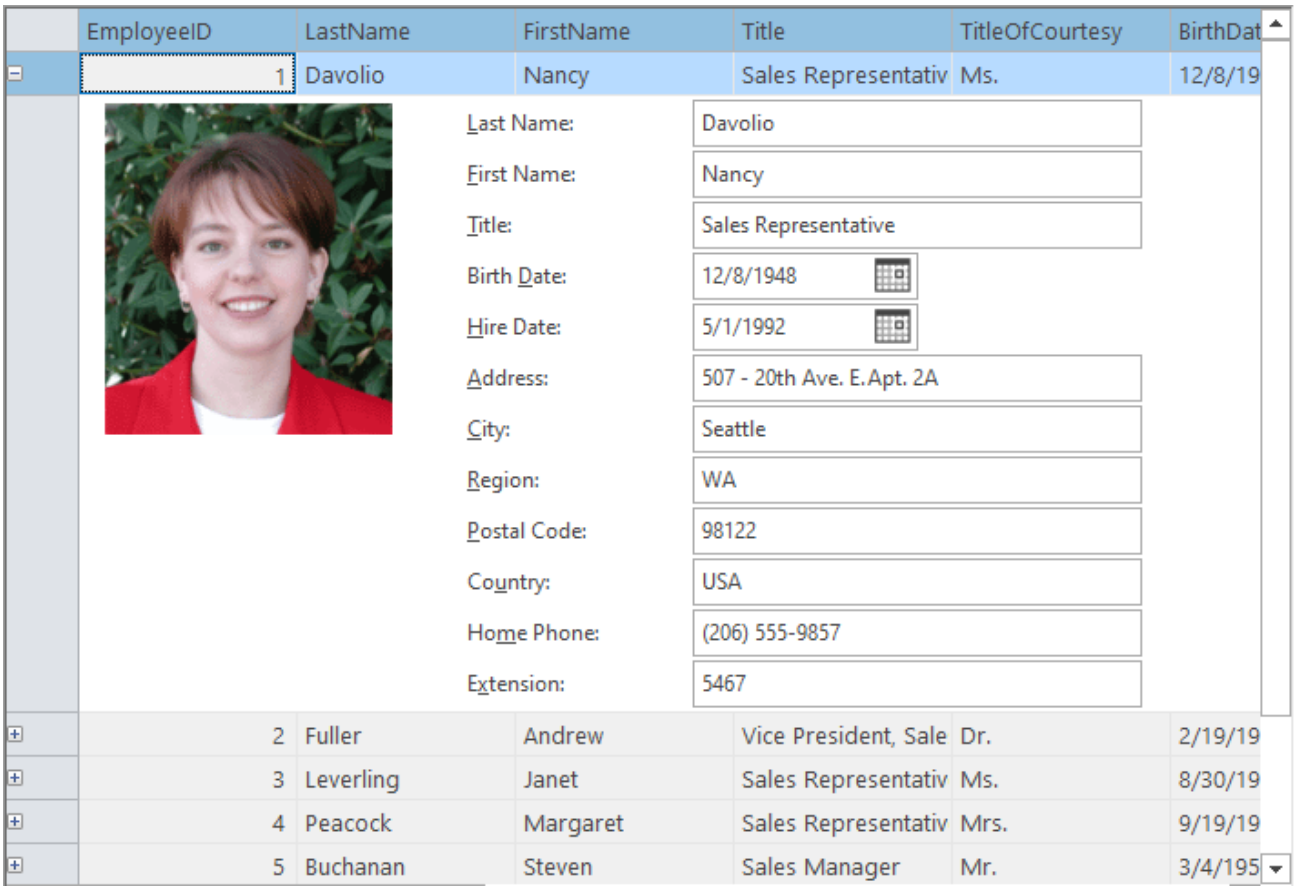
```
// ヘッダーテキストのフォントを設定します
c1FlexGrid1.Styles["Fixed"].Font = new Font("Tahoma", 10, FontStyle.Bold);

// ヘッダーテキストの前色を設定します
c1FlexGrid1.Styles["Fixed"].ForeColor = Color.PaleVioletRed;

// ヘッダーテキストの背景色を設定します
c1FlexGrid1.Styles["Fixed"].BackColor = Color.LemonChiffon;

// ヘッダーテキストにテキスト効果を適用します
c1FlexGrid1.Styles["Fixed"].TextEffect = C1.Win.C1FlexGrid.TextEffectEnum.Inset;
```

VB.NET

```
' ヘッダーテキストのフォントを設定します
c1FlexGrid1.Styles("Fixed").Font = New Font("Tahoma", 10, FontStyle.Bold)

' ヘッダーテキストの前色を設定します
c1FlexGrid1.Styles("Fixed").ForeColor = Color.PaleVioletRed

' ヘッダーテキストの背景色を設定します
c1FlexGrid1.Styles("Fixed").BackColor = Color.LemonChiffon

' ヘッダーテキストにテキスト効果を適用します
c1FlexGrid1.Styles("Fixed").TextEffect = C1.Win.C1FlexGrid.TextEffectEnum.Inset
```

行詳細

行詳細は、行(レコード)に関連する追加情報を別の展開可能なレイヤの形で提供する機能です。この場合、最初のレイヤである行には基本情報が含まれ、2 番目のレイヤには詳細な情報が含まれます。この機能は、追加情報が多過ぎて使用可能な画面内に表示できない場合や、レコードごとの追加情報に一貫性がない場合に特に便利です。



FlexGrid は、IC1FlexGridRowDetail インタフェースを使用して行詳細機能を提供します。詳細行に置かれる詳細コントロールは、このインタフェースを実装します。また、FlexGrid は、InputPanel および FlexGrid というテンプレートユーザーコントロールを独立したアセンブリ **C1.Win.C1FlexGrid.RowDetails.4.5.2.dll** で提供しており、これらのコントロールを詳細行ですぐに使用できます。FlexGrid の任意の行に詳細セクションを追加でき、データを 1 つのテンプレートにグループ化してオプションで表示することができます。これにより、ユーザーは、行を選択するだけで、その行に関連する追加データを表示できます。また、グリッドには、組み込みの展開/折りたたみボタンも用意されており、展開可能な行内のデータの表示/非表示を制御できます。

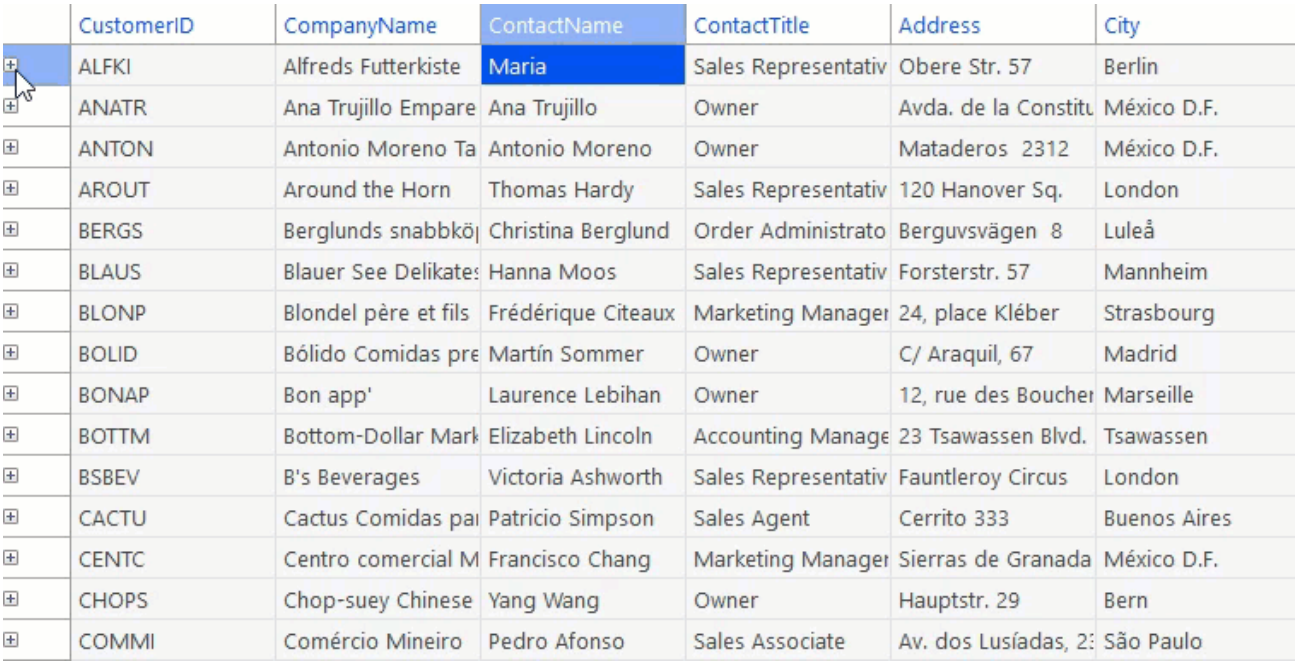
行詳細機能がよく使用されるシナリオには、次のようなものがあります。

1. **フォーム内編集**: InputPanel コントロールを置くことで、ユーザー入力を取得し、レコードに情報を挿入できるようにします。
2. **階層グリッド**: マスターグリッドと、レコードに関する追加情報を表示する詳細グリッドから成ります。
3. **カスタム行詳細**: 任意のコントロールを使用して行詳細テンプレートを作成できます。

これらのシナリオの実装方法について、以下のセクションで説明します。

フォーム内編集

FlexGrid は、詳細行に InputPanel コントロールを置くことで、フォーム内編集をサポートします。InputPanel コントロールには、フォームと同様に、ユーザーが値を入力したり編集することができるデータフィールドが含まれます。ユーザーがフィールドの編集を終了すると、値が選択した行に反映されます。



	CustomerID	CompanyName	ContactName	ContactTitle	Address	City
⊞	ALFKI	Alfreds Futterkiste	Maria	Sales Representativ	Obere Str. 57	Berlin
⊞	ANATR	Ana Trujillo Empare	Ana Trujillo	Owner	Avda. de la Constitu	México D.F.
⊞	ANTON	Antonio Moreno Ta	Antonio Moreno	Owner	Mataderos 2312	México D.F.
⊞	AROUT	Around the Horn	Thomas Hardy	Sales Representativ	120 Hanover Sq.	London
⊞	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrato	Berguvsvägen 8	Luleå
⊞	BLAUS	Blauer See Delikates	Hanna Moos	Sales Representativ	Forsterstr. 57	Mannheim
⊞	BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg
⊞	BOLID	Bólido Comidas pre	Martín Sommer	Owner	C/ Araquil, 67	Madrid
⊞	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Boucher	Marseille
⊞	BOTTM	Bottom-Dollar Mark	Elizabeth Lincoln	Accounting Manage	23 Tsawassen Blvd.	Tsawassen
⊞	BSBEV	B's Beverages	Victoria Ashworth	Sales Representativ	Fauntleroy Circus	London
⊞	CACTU	Cactus Comidas par	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires
⊞	CENTC	Centro comercial M	Francisco Chang	Marketing Manager	Sierras de Granada	México D.F.
⊞	CHOPS	Chop-suey Chinese	Yang Wang	Owner	Hauptstr. 29	Bern
⊞	COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate	Av. dos Lusíadas, 23	São Paulo

FlexGrid でフォーム内編集を実装するには、**C1.Win.C1FlexGrid.RowDetails.4.5.2.dll** への参照を追加し、**IC1FlexGridRowDetail** インタフェースを実装済みの C1InputPanelRowDetail クラスを使用します。次に、このクラスのインスタンスを C1FlexGrid クラスの **RowDetailProvider** プロパティに割り当てます。これにより、InputPanel コントロールが詳細行に追加され、フォーム内編集が可能になります。

C#

```
flexGrid.RowDetailProvider = (g, r) => new C1InputPanelRowDetail();
flexGrid.RowDetailsVisibilityMode = RowDetailsVisibilityMode.VisibleWhenSelected;
```

VB.NET

```
flexGrid.RowDetailProvider = Function(g, r) New C1InputPanelRowDetail()
flexGrid.RowDetailsVisibilityMode = RowDetailsVisibilityMode.VisibleWhenSelected
```

階層ビュー

階層ビューは、マスター/詳細モデルのことです。最上位レベルのグリッドを「マスターグリッド」、グリッド内グリッドを「詳細グリッド」と言います。詳細グリッドには、マスターグリッドで展開された行に関する追加情報が表示されます。たとえば、次の例で示されているマスター/詳細構造では、*Customer* テーブルと *Order* テーブルが使用されています。これらのテーブルはどちらも、関係を定義する共通のデータ要素として *CustomerID* を持っています。

	CustomerID	CompanyName	ContactName	ContactTitle	Address	City
⊞	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representativ	Obere Str. 57	Berlin
⊞	ANATR	Ana Trujillo Empare	Ana Trujillo	Owner	Avda. de la Constitu	México D.F.
⊞	ANTON	Antonio Moreno Ta	Antonio Moreno	Owner	Mataderos 2312	México D.F.
⊞	AROUT	Around the Horn	Thomas Hardy	Sales Representativ	120 Hanover Sq.	London
⊞	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrato	Berguvsvägen 8	Luleå
⊞	BLAUS	Blauer See Delikates	Hanna Moos	Sales Representativ	Forsterstr. 57	Mannheim
⊞	BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg
⊞	BOLID	Bólido Comidas pre	Martín Sommer	Owner	C/ Araquil, 67	Madrid
⊞	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Boucher	Marseille
⊞	BOTTM	Bottom-Dollar Mark	Elizabeth Lincoln	Accounting Manage	23 Tsawassen Blvd.	Tsawassen
⊞	BSBEV	B's Beverages	Victoria Ashworth	Sales Representativ	Fauntleroy Circus	London
⊞	CACTU	Cactus Comidas par	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires
⊞	CENTC	Centro comercial M	Francisco Chang	Marketing Manager	Sierras de Granada	México D.F.
⊞	CHOPS	Chop-suey Chinese	Yang Wang	Owner	Hauptstr. 29	Bern
⊞	COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate	Av. dos Lusíadas, 23	São Paulo

FlexGrid で階層グリッドを実装するには、**C1.Win.C1FlexGrid.RowDetails.4.5.2.dll** への参照を追加し、**IC1FlexGridRowDetail** インタフェースを実装済みの C1FlexGridRowDetail クラスを使用します。次に、このクラスのインスタンスを **C1FlexGrid** クラスの **RowDetailProvider** プロパティに割り当てて、詳細行に別のグリッドをネストします。これで、階層グリッドインタフェースを作成できます。

C#

```
private void FlexGrid_Load(object sender, EventArgs e)
        {
            string conn = Util.GetConnectionString();
            var ds = new DataSet();
            string[] tables = "Customers, Orders".Split(',');
            foreach (string tableName in tables)
            {
                Util.FillTable(ds, tableName, conn);
            }

          // マスターグリッドと詳細グリッド間の関係を定義します
            ds.Relations.Add("Customers_Orders",
                ds.Tables["Customers"].Columns["CustomerID"],
                ds.Tables["Orders"].Columns["CustomerID"]);
            flexGrid.DataSource = ds;
            flexGrid.DataMember = "Customers";
            flexGrid.RowDetailProvider = (g, r) => new C1FlexGridRowDetail();
            flexGrid.AreRowDetailsFrozen = false;
        }
    }
```

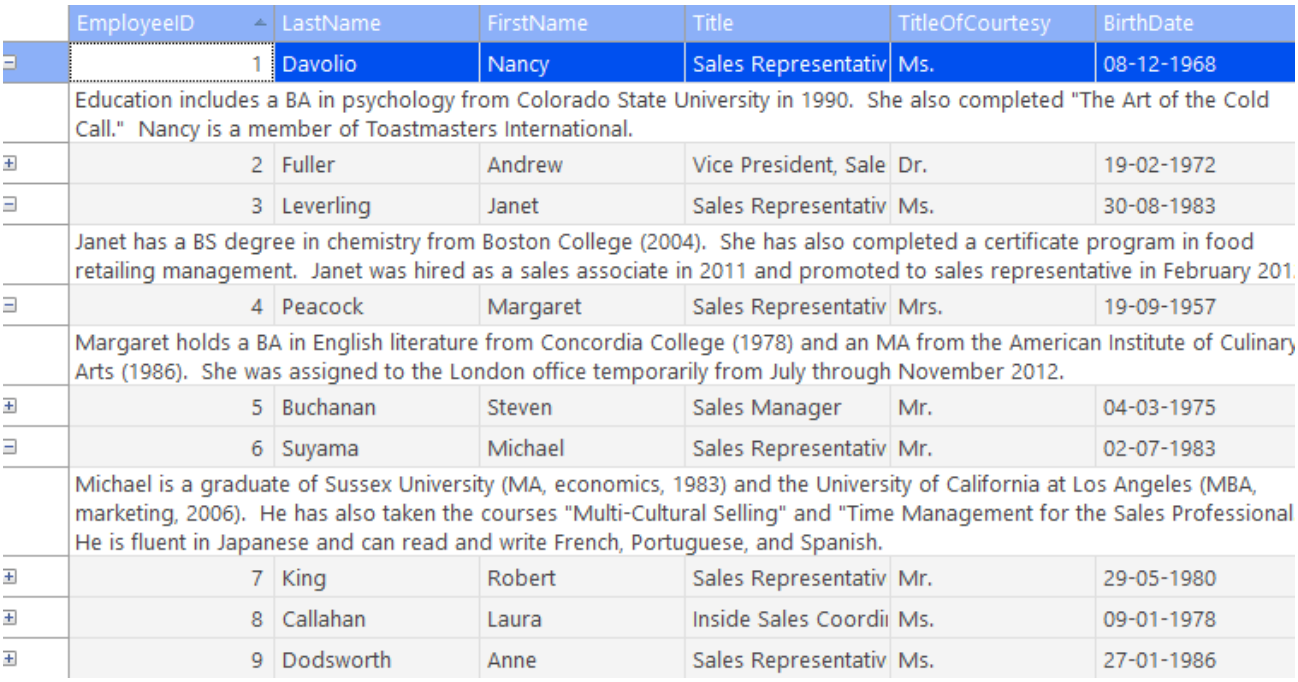
VB.NET

```
Private Sub FlexGrid_Load(ByVal sender As Object, ByVal e As EventArgs)
      Dim conn As String = Util.GetConnectionString()
      Dim ds = New DataSet()
      Dim tables As String() = "Customers, Orders".Split(","c)

      For Each tableName As String In tables
          Util.FillTable(ds, tableName, conn)
      Next
      ' マスターグリッドと詳細グリッド間の関係を定義します
      ds.Relations.Add("Customers_Orders", ds.Tables("Customers").Columns("CustomerID"),
ds.Tables("Orders").Columns("CustomerID"))
```

```
        flexGrid.DataSource = ds
        flexGrid.DataMember = "Customers"
        flexGrid.RowDetailProvider = Function(g, r) New C1FlexGridRowDetail()
        flexGrid.AreRowDetailsFrozen = False
    End Sub
```

カスタム行詳細

グリッドの詳細行には、InputPanel コントロールと FlexGrid コントロールのほかに、カスタムコントロールを置くこともできます。たとえば、次の例では、テキストラベルコントロールが行にアタッチされ、グリッドのサイズに影響を与えることなく追加情報を得ています。



EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate
1	Davolio	Nancy	Sales Representativ	Ms.	08-12-1968
Education includes a BA in psychology from Colorado State University in 1990. She also completed "The Art of the Cold Call." Nancy is a member of Toastmasters International.					
2	Fuller	Andrew	Vice President, Sale	Dr.	19-02-1972
3	Leverling	Janet	Sales Representativ	Ms.	30-08-1983
Janet has a BS degree in chemistry from Boston College (2004). She has also completed a certificate program in food retailing management. Janet was hired as a sales associate in 2011 and promoted to sales representative in February 201					
4	Peacock	Margaret	Sales Representativ	Mrs.	19-09-1957
Margaret holds a BA in English literature from Concordia College (1978) and an MA from the American Institute of Culinary Arts (1986). She was assigned to the London office temporarily from July through November 2012.					
5	Buchanan	Steven	Sales Manager	Mr.	04-03-1975
6	Suyama	Michael	Sales Representativ	Mr.	02-07-1983
Michael is a graduate of Sussex University (MA, economics, 1983) and the University of California at Los Angeles (MBA, marketing, 2006). He has also taken the courses "Multi-Cultural Selling" and "Time Management for the Sales Professional He is fluent in Japanese and can read and write French, Portuguese, and Spanish.					
7	King	Robert	Sales Representativ	Mr.	29-05-1980
8	Callahan	Laura	Inside Sales Coordi	Ms.	09-01-1978
9	Dodsworth	Anne	Sales Representativ	Ms.	27-01-1986

FlexGrid でカスタム行詳細を実装するには、IC1FlexGridRowDetail インタフェースを実装するユーザーコントロールを作成する必要があります。たとえば、次のコードでは、詳細行に置かれるテキストラベルコントロールを表す **CustomRowDetail** というクラスを作成しています。次に、このクラスのオブジェクトを **C1FlexGrid** クラスの **RowDetailProvider** プロパティに割り当てて、カスタムコントロールが詳細行に追加情報を表示できるようにしています。

C#

```
private void CustomSample_Load(object sender, EventArgs e)
{
   flexGrid.DataSource = DemoDataSource("Employees");
   flexGrid.RowDetailProvider = (g, r) => new CustomRowDetail();
   flexGrid.RowDetailsVisibilityMode = RowDetailsVisibilityMode.VisibleWhenSelected;
   flexGrid.Cols["Notes"].Visible = false;
}

 // 従業員に関するメモ付きのラベルを表示するカスタム行詳細クラス
 public class CustomRowDetail : C1Label, IC1FlexGridRowDetail
  {

    // FlexGrid詳細コントロールを表示する前にコントロールを設定するために使用されます
    void IC1FlexGridRowDetail.Setup(C1FlexGrid parentGrid, int rowIndex)
      {
          var bs = new BindingSource(parentGrid.DataSource, parentGrid.DataMember);
          bs.Position = parentGrid.Rows[rowIndex].DataIndex;
          DataField = "Notes";
          DataSource = bs;
      }
```

```
        // FlexGrid詳細コントロールのサイズを更新するために使用されます
        void IC1FlexGridRowDetail.UpdateSize(C1FlexGrid parentGrid, int rowIndex, Size
proposedSize)
        {
            var srSz = parentGrid.ScrollableRectangle.Size;
            var sz = TextRenderer.MeasureText(Text, Font, srSz,
TextFormatFlags.WordBreak);
            sz.Width = Math.Max(sz.Width, srSz.Width);
            Size = sz;
        }
    }
```

VB.NET

```
    Private Sub CustomSample_Load(ByVal sender As Object, ByVal e As EventArgs)
        flexGrid.DataSource = DemoDataSource("Employees")
        flexGrid.RowDetailProvider = Function(g, r) New CustomRowDetail()
        flexGrid.RowDetailsVisibilityMode = RowDetailsVisibilityMode.VisibleWhenSelected
        flexGrid.Cols("Notes").Visible = False
    End Sub


    ' 従業員に関するメモ付きのラベルを表示するカスタム行詳細クラス
    Public Class CustomRowDetail
        Inherits C1Label
        Implements IC1FlexGridRowDetail


        ' FlexGrid詳細コントロールを表示する前にコントロールを設定するために使用されます
        Private Sub Setup(ByVal parentGrid As C1FlexGrid, ByVal rowIndex As Integer)
            Dim bs = New BindingSource(parentGrid.DataSource, parentGrid.DataMember)
            bs.Position = parentGrid.Rows(rowIndex).DataIndex
            DataField = "Notes"
            DataSource = bs
        End Sub


        ' FlexGrid詳細コントロールのサイズを更新するために使用されます
        Private Sub UpdateSize(ByVal parentGrid As C1FlexGrid, ByVal rowIndex As Integer,
ByVal proposedSize As Size)
            Dim srSz = parentGrid.ScrollableRectangle.Size
            Dim sz = TextRenderer.MeasureText(Text, Font, srSz,
TextFormatFlags.WordBreak)
            sz.Width = Math.Max(sz.Width, srSz.Width)
            Size = sz
        End Sub
    End Class
```

サイズ変更

行の高さの設定

FlexGrid では、RowCollection クラスの DefaultSize プロパティを使用して、グリッド全体の行の高さを設定できます。Row クラスの **Height** プロパティを設定して、特定の行の高さを指定することもできます。**Height** プロパティのデフォルト値は -1 です。これは、行が **DefaultSize** プロパティによって指定される高さになることを意味しています。

以下のコードを使用して、WinForms FlexGrid の行のデフォルトの高さを設定します。

C#

```
//すべての行のデフォルトサイズを設定します
c1FlexGrid1.Rows.DefaultSize = 50;
```

```
//特定の行の高さを設定します
c1FlexGrid1.Rows[1].Height = 55;
```

VB.NET

```
' すべての行のデフォルトサイズを設定します
c1FlexGrid1.Rows.DefaultSize = 50

' 特定の行の高さを設定します
c1FlexGrid1.Rows(1).Height = 55
```

行の高さの自動調整

テキストの長さと文字の折り返しオプションに基づいて行の高さを調整するために、FlexGrid には、AutoSizeRow() メソッドと **AutoSizeRows()** メソッドが用意されています。**AutoSizeRow()** メソッドは、指定された行の高さを自動的に調整します。一方、**AutoSizeRows()** メソッドはセル範囲に対して使用されます。

以下のコードは、WinForms FlexGrid でテキスト長に基づいて行の高さを自動的に調整する方法を示しています。

C#

```
//4行目の高さを自動的に調整します
c1FlexGrid1.AutoSizeRow(4);

//すべての行の高さを自動的に調整します
c1FlexGrid1.AutoSizeRows();

//セル範囲の高さを自動的に調整します
//c1FlexGrid1.AutoSizeRows(2, 4, 5, 6, 10, C1.Win.C1FlexGrid.AutoSizeFlags.None);
```

VB.NET

```
'4行目の高さを自動的に調整します
c1FlexGrid1.AutoSizeRow(4)

'すべての行の高さを自動的に調整します
c1FlexGrid1.AutoSizeRows()

セル範囲の高さを自動的に調整します
'c1FlexGrid1.AutoSizeRows(2, 4, 5, 6, 10, C1.Win.C1FlexGrid.AutoSizeFlags.None)
```

行の最小/最大高さの設定

FlexGrid では、RowCollection クラスの MinSize プロパティと MaxSize プロパティを使用して、行の高さの範囲を設定できます。この機能は、AllowResizing プロパティに **true** を設定した場合、または **AutoSizeRow()** または **AutoSizeRows()** メソッドを使用する場合などに特に便利です。

以下のコードを使用して、WinForms FlexGrid で行の高さの範囲を指定します。

C#

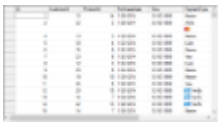
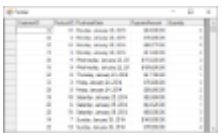
```
//行の最大高さを設定します
c1FlexGrid1.Rows.MaxSize = 50;

//行の最小の高さを設定します
c1FlexGrid1.Rows.MinSize = 20;
```

VB.NET

```
'行の最大高さを設定します
c1FlexGrid1.Rows.MaxSize = 50

'行の最小の高さを設定します
 c1FlexGrid1.Rows.MinSize = 20
```

セル

セルは、グリッドの最小単位です。ほとんどのシナリオでは、行または列レベルの作業が便利ですが、セルレベルで実行する必要がある操作もあります。たとえば、データの取得、設定、削除がそうです。

トピック	スナップショット	コンテンツ
基本的な操作		セルに関連する基本操作を実行する方法について説明します。 • セルに値を設定 • セル範囲に値を設定 • セル(範囲)から値をクリア • セルに画像を設定 • セルにツールチップを表示 • セル値の取得
セルの書式設定		さまざまなシナリオでセルやセルデータの書式を設定する方法について説明します。 • セルコンテンツ • セルの外観 • 条件付き書式設定 • オーナー描画セル

基本的な操作

セルに値を設定

FlexGrid でセルに値を設定する方法は 2 つあります。**Item** プロパティ(インデクサ)または C1FlexGrid クラスの **SetData** メソッドを使用できます。

以下のコードを使用して、WinForms FlexGrid のセルに値を設定します。

C#

```
// インデックスを使用してデータを設定します
c1FlexGrid1[2, 3] = "2nd col 3rd row";

// SetDataメソッドを使用してデータを設定します
c1FlexGrid1.SetData(2, 4, "2nd col 4th row");
```

VB.NET

```
' インデックスを使用してデータを設定します
c1FlexGrid1(2, 3) = "2nd col 3rd row"

' SetDataメソッドを使用してデータを設定します
c1FlexGrid1.SetData(2, 4, "2nd col 4th row")
```

セル範囲に値を設定

セル範囲に値を設定するには、CellRange クラスの **Data** プロパティまたは **C1FlexGrid** クラスの **SetData** メソッドを使用できます。

以下のコードは、WinForms FlexGrid でセル範囲に値を設定する方法を示しています。

C#

```
// セル範囲を取得します
C1.Win.C1FlexGrid.CellRange cr = c1FlexGrid1.GetCellRange(2, 3, 5, 6);

// 方法 1: dataプロパティを使用して、セル範囲にデータを設定します
cr.Data = "Cell Range";

// 方法 2: SetDataメソッドを使用してセル範囲にデータを設定します
// c1FlexGrid1.SetData(cr, "Cell Range");
```

VB.NET

```
' セル範囲を取得します
Dim cr As C1.Win.C1FlexGrid.CellRange = c1FlexGrid1.GetCellRange(2, 3, 5, 6)

' dataプロパティを使用して、セル範囲にデータを設定します
cr.Data = "Cell Range"

' SetDataメソッドを使用してセル範囲にデータを設定します
' c1FlexGrid1.SetData(cr, "Cell Range")
```

セル(範囲)から値をクリア

セルまたはセル範囲のコンテンツをクリアするには、2 つの方法があります。プログラムからは、インデクサまたは **SetData** メソッドを使用してセルのコンテンツに空の文字列を設定することで、コンテンツをクリアできます。または、**AutoClipboard** プロパティに **true** を設定することで、ユーザーが[Del]キーを押して値を削除できるようにします。

以下のコードは、キーボード操作によって WinForms FlexGrid のセルから値をクリアできるようにする方法を具体的に示しています。

C#

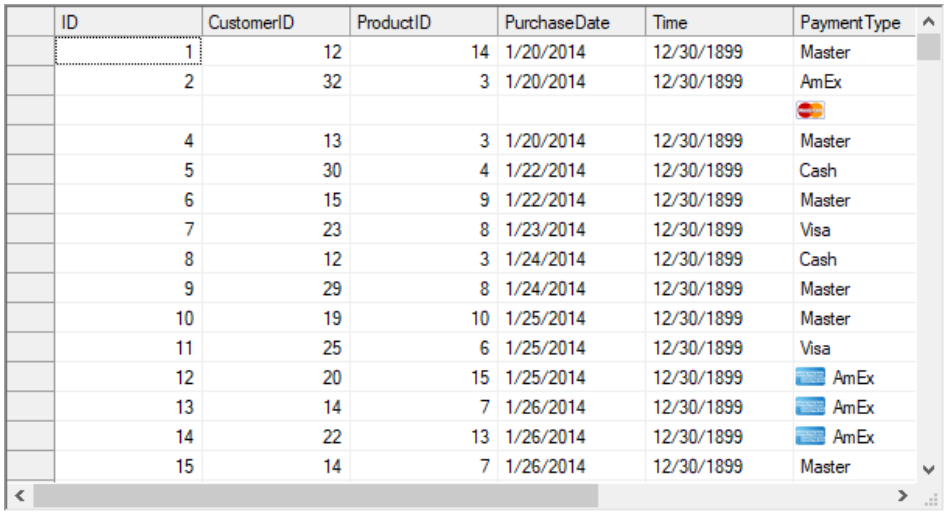
```
// ユーザーがDeleteキーを押してセルの内容をクリアするなどのキーボード操作を実行できるようにします
c1FlexGrid1.AutoClipboard = true;

// コードを使用して特定のセルのデータをクリアします
c1FlexGrid1.SetData(3, 4, "");
```

VB.NET

```
' ユーザーがDeleteキーを押してセルの内容をクリアするなどのキーボード操作を実行できるようにします
c1FlexGrid1.AutoClipboard = True

' コードを使用して特定のセルのデータをクリアします
c1FlexGrid1.SetData(3, 4, "")
```

セルに画像を設定

セルに画像を設定するには、**C1FlexGrid** クラスの **SetCellImage** メソッドを使用できます。また、**CellRange** クラスの **Image** プロパティを使用してセル範囲に画像を設定することもできます。デフォルトでは、セルにテキストと画像の両方が表示されます。ただし、**ImageAndText** プロパティに **false** を設定することで、画像のみを表示するように選択できます。



	ID	CustomerID	ProductID	PurchaseDate	Time	PaymentType
	1	12	14	1/20/2014	12/30/1899	Master
	2	32	3	1/20/2014	12/30/1899	AmEx
	4	13	3	1/20/2014	12/30/1899	Master
	5	30	4	1/22/2014	12/30/1899	Cash
	6	15	9	1/22/2014	12/30/1899	Master
	7	23	8	1/23/2014	12/30/1899	Visa
	8	12	3	1/24/2014	12/30/1899	Cash
	9	29	8	1/24/2014	12/30/1899	Master
	10	19	10	1/25/2014	12/30/1899	Master
	11	25	6	1/25/2014	12/30/1899	Visa
	12	20	15	1/25/2014	12/30/1899	AmEx
	13	14	7	1/26/2014	12/30/1899	AmEx
	14	22	13	1/26/2014	12/30/1899	AmEx
	15	14	7	1/26/2014	12/30/1899	Master

以下のコードを使用して、WinForms FlexGrid のセルに画像を設定します。

C#

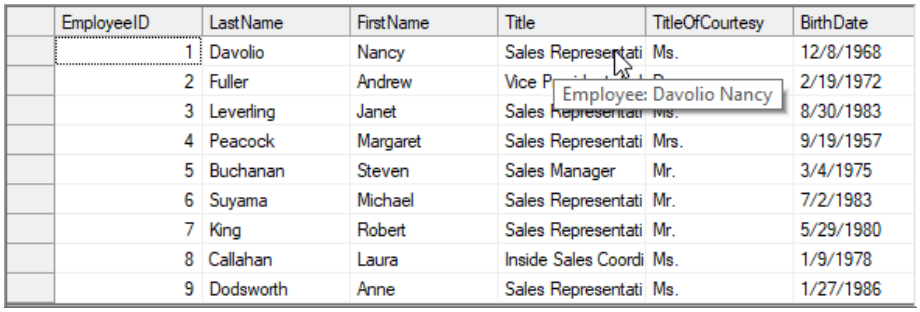
```
// セル(3,6)に画像を設定します
c1FlexGrid1.SetCellImage(3, 6, Image.FromFile("master.png"));

// セル範囲(12,6)から(14、6)に画像を設定します
C1.Win.C1FlexGrid.CellRange cr;
cr = c1FlexGrid1.GetCellRange(12, 6, 14, 6);
cr.Image = Image.FromFile("amex.jpg");

// テキストなしで画像を表示します
c1FlexGrid1.Rows[3].ImageAndText = false;
```

VB.NET

```
' セル(3,6)に画像を設定します
c1FlexGrid1.SetCellImage(3, 6, Image.FromFile("master.png"))

' セル範囲(12,6)から(14、6)に画像を設定します
Dim cr As C1.Win.C1FlexGrid.CellRange
cr = c1FlexGrid1.GetCellRange(12, 6, 14, 6)
cr.Image = Image.FromFile("amex.jpg")

' テキストなしで画像を表示します
c1FlexGrid1.Rows(3).ImageAndText = False
```

セルにツールチップを表示

部分的に非表示のセルのコンテンツをツールチップとして表示するために、FlexGrid には、**C1FlexGrid** クラスの **ShowCellLabels** プロパティがあります。また、**MouseEnterCell** イベントと **MouseLeaveCell** イベントを使用して、ツールチップの形式で追加情報を表示することもできます。



以下のコードは、WinForms FlexGrid のセルにツールチップを表示する方法を示しています。

C#

```
private void Form1_Load(object sender, EventArgs e)
        {
            // 'c1NWindDataSet.Employees'テーブルにデータをロードします。 必要に応じて、移動または削除できます。
            this.employeesTableAdapter.Fill(this.c1NWindDataSet.Employees);

            for (int i = c1FlexGrid1.Rows.Fixed; i < c1FlexGrid1.Rows.Count; i++)
            {
                c1FlexGrid1.Rows[i].UserData = "Employee: " + c1FlexGrid1[i, 2] + " " + c1FlexGrid1[i, 3];
            }
        }
        private void C1FlexGrid1_MouseEnterCell(object sender, C1.Win.C1FlexGrid.RowColEventArgs e)
        {
            if (e.Row >= c1FlexGrid1.Rows.Fixed)
            {
                string tip;
                tip = c1FlexGrid1.Rows[e.Row].UserData.ToString();
                // ツールチップを表示します
                toolTip1.SetToolTip(c1FlexGrid1, tip);
            }
        }

        private void C1FlexGrid1_MouseLeaveCell(object sender, C1.Win.C1FlexGrid.RowColEventArgs e)
        {
            // ツールチップを非表示にします
            toolTip1.SetToolTip(c1FlexGrid1, "");
        }
```

VB.NET

```
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
        ' 'c1NWindDataSet.Employees'テーブルにデータをロードします。 必要に応じて、移動または削除できます。
        Me.employeesTableAdapter.Fill(Me.c1NWindDataSet.Employees)

        For i As Integer = c1FlexGrid1.Rows.Fixed To c1FlexGrid1.Rows.Count - 1
            c1FlexGrid1.Rows(i).UserData = "Employee: " & c1FlexGrid1(i, 2) & " " + c1FlexGrid1(i, 3)
        Next
    End Sub

    Private Sub C1FlexGrid1_MouseEnterCell(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs)
        If e.Row >= c1FlexGrid1.Rows.Fixed Then
            Dim tip As String
            tip = c1FlexGrid1.Rows(e.Row).UserData.ToString()
```

```
            ' ツールチップを表示します
            toolTip1.SetToolTip(c1FlexGrid1, tip)
        End If
    End Sub

    Private Sub C1FlexGrid1_MouseLeaveCell(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs)
        ' ツールチップを非表示にします
        toolTip1.SetToolTip(c1FlexGrid1, "")
    End Sub
```

セル値の取得

FlexGrid のセル値を取得する方法は、要件に応じて数多くあります。以下の表では、単一のセルまたはセル範囲から生データまたは書式設定されたデータを取得する方法など、いくつかのシナリオについて説明します。

必要条件	メソッド/プロパティ	使用方法

生データの取得	**Item** プロパティ（インデクサ）	ExampleCode
		`var data = c1FlexGrid1[1, 1];`
		`System.Diagnostics.Debug.WriteLine($"セルデータ: {data}");`

	GetData() メソッド	ExampleCode
		`var data1 = c1FlexGrid1.GetData(1, 1);`
		`System.Diagnostics.Debug.WriteLine($"セルデータ: {data1}");`

書式設定されたデータの取得	**GetDataDisplay()** メソッド	ExampleCode
		`var data2 = c1FlexGrid1.GetDataDisplay(1, 1);`
		`System.Diagnostics.Debug.WriteLine($"表示データ: {data2}");`

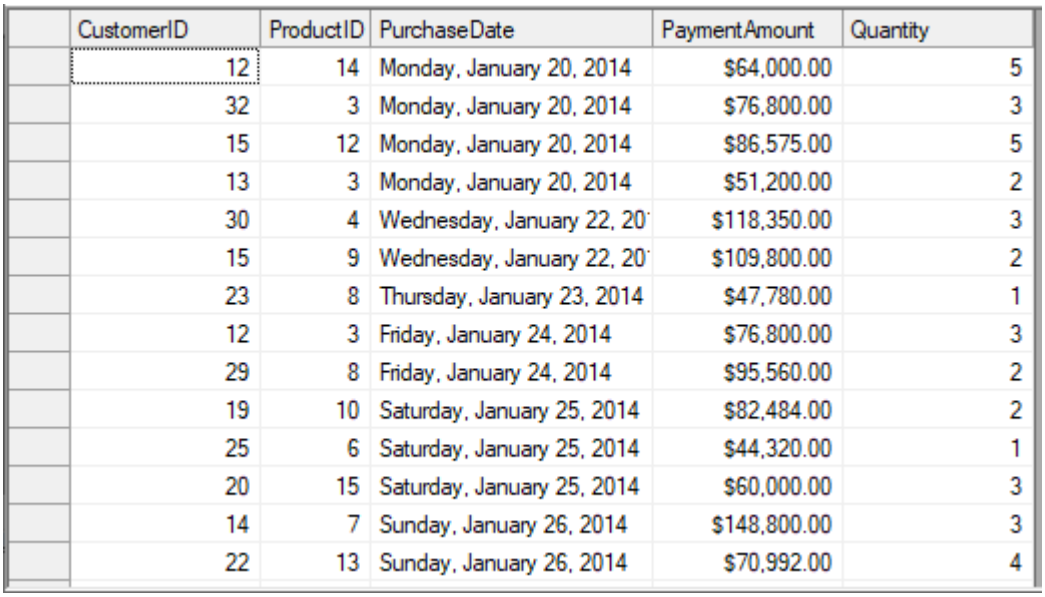
セル範囲の値の取得	**Clip** プロパティ	ExampleCode
		`var data3 = c1FlexGrid1.Clip;`
		`System.Diagnostics.Debug.WriteLine($"クリップデータ: {data3}");`

	GetCellRange メソッド	ExampleCode
		`var data4 = c1FlexGrid1.GetCellRange(1, 1);`
		`System.Diagnostics.Debug.WriteLine($"セル範囲データ: {data4.Clip}");`

セルの書式設定

セルコンテンツ

セルのコンテンツを書式設定および表示する方法を制御するには、Row オブジェクトまたは Column オブジェクトの Format プロパティを設定して、.NET Framework の **String.Format** プロパティで使用される書式設定と同様に文字列を書式設定します。たとえば、以下のサンプルコードは、3 番目の列を Date、4 番目の列を Currency として書式設定します。



	CustomerID	ProductID	PurchaseDate	PaymentAmount	Quantity
	12	14	Monday, January 20, 2014	$64,000.00	5
	32	3	Monday, January 20, 2014	$76,800.00	3
	15	12	Monday, January 20, 2014	$86,575.00	5
	13	3	Monday, January 20, 2014	$51,200.00	2
	30	4	Wednesday, January 22, 20	$118,350.00	3
	15	9	Wednesday, January 22, 20	$109,800.00	2
	23	8	Thursday, January 23, 2014	$47,780.00	1
	12	3	Friday, January 24, 2014	$76,800.00	3
	29	8	Friday, January 24, 2014	$95,560.00	2
	19	10	Saturday, January 25, 2014	$82,484.00	2
	25	6	Saturday, January 25, 2014	$44,320.00	1
	20	15	Saturday, January 25, 2014	$60,000.00	3
	14	7	Sunday, January 26, 2014	$148,800.00	3
	22	13	Sunday, January 26, 2014	$70,992.00	4

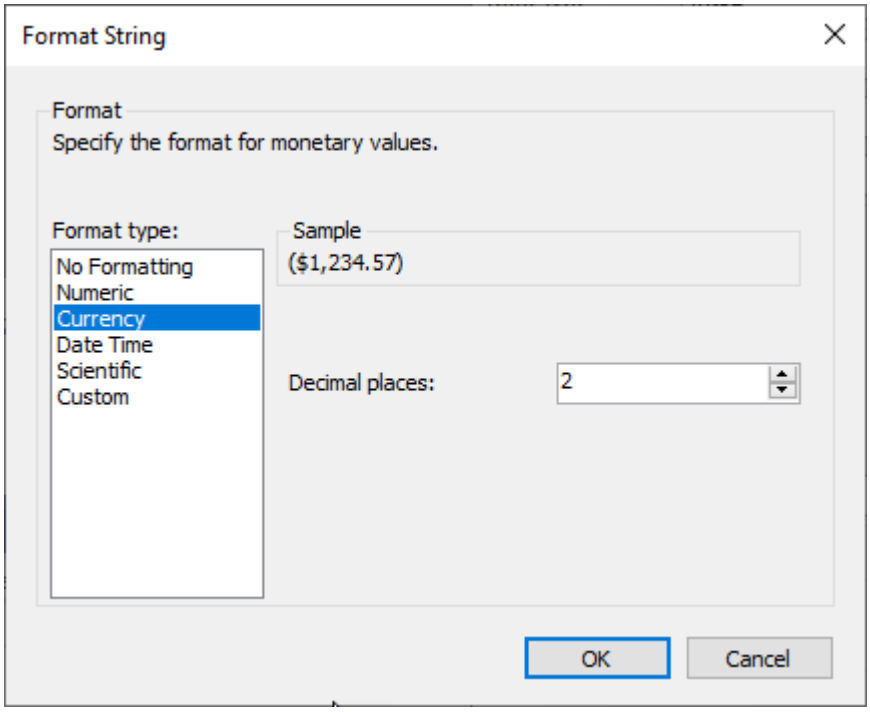
C#

```
//短い日付書式を設定します
c1FlexGrid1.Cols[3].Format = "D";

//通貨書式を設定します
c1FlexGrid1.Cols[4].Format = "c";
```

VB.NET

```
'短い日付書式を設定します
c1FlexGrid1.Cols(3).Format = "D"

'通貨書式を設定します
c1FlexGrid1.Cols(4).Format = "c"
```

また、FlexGrid は、**Format** プロパティを設定するための設計時オプションとして**[書式文字列]**ダイアログを提供します。[書式文字列]ダイアログには、**列タスク**メニューの**[書式文字列]**フィールドにある**省略符**ボタンをクリックしてアクセスできます。または、**C1FlexGrid 列エディタ**で Format プロパティを使用することもできます。



[書式文字列]ダイアログは各列に固有です。選択した列の **Format** プロパティのみが変更されます。

セルの外観

FlexGrid は、配置、フォント、色、境界線などのセルの外観を処理するために CellStyle オブジェクトを提供しています。グリッドには、グリッドの書式設定に使用されるスタイルのコレクションを保持する **Styles** プロパティがあります。このコレクションには、固定セル、スクロール可能セル、選択範囲、フォーカスセルなど、グリッド要素の外観を定義する組み込みメンバがいくつか含まれます。これらのスタイルを変更して、グリッドの外観を修正できます。

CustomerID	ProductID	PurchaseDate	PaymentAmount	Quantity
12	14	1/20/2014	$64,000.00	5
32	3	1/20/2014	$76,800.00	3
15	12	1/20/2014	$86,575.00	5
13	3	1/20/2014	$51,200.00	2
30	4	1/22/2014	$118,350.00	3
15	9	1/22/2014	$109,800.00	2
23	8	1/23/2014	$47,780.00	1
12	3	1/24/2014	$76,800.00	3
29	8	1/24/2014	$95,560.00	2
19	10	1/25/2014	$82,484.00	2
25	6	1/25/2014	$44,320.00	1
20	15	1/25/2014	$60,000.00	3
14	7	1/26/2014	$148,800.00	3
22	13	1/26/2014	$70,992.00	4

組み込みスタイルを変更してグリッドの外観を変更する方法が最も簡単ですが、独自のカスタムスタイルを作成して、それをセル、行、列に割り当てることもできます。

WinForms FlexGrid のセルの外観を変更するには、次のコードを使用します。

C#

```
//組み込みのスタイルをカスタマイズします
CellStyle cs = c1FlexGrid1.Styles.Focus;
```

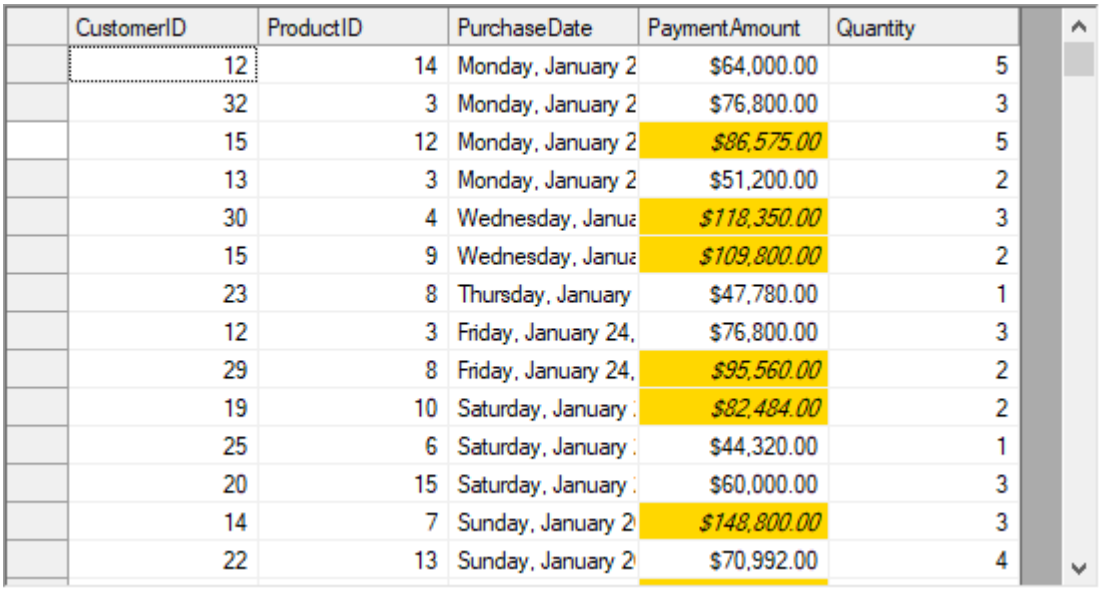
```
cs.Font = new Font(c1FlexGrid1.Font, FontStyle.Bold);
cs.ForeColor = Color.Green;
cs.BackColor = Color.Red;

//カスタムスタイルを作成します
CellStyle cs1 = c1FlexGrid1.Styles.Add("NewStyle");
cs1.BackColor = Color.Aqua;
cs1.ForeColor = Color.Blue;

//カスタムスタイルを割り当てます
c1FlexGrid1.Cols[3].Style = cs1;
```

VB.NET

```
Dim cs As CellStyle = c1FlexGrid1.Styles.Focus
    cs.Font = New Font(c1FlexGrid1.Font, FontStyle.Bold)
    cs.ForeColor = Color.Green
    cs.BackColor = Color.Red

    Dim cs1 As CellStyle = c1FlexGrid1.Styles.Add("NewStyle")
    cs1.BackColor = Color.Aqua
    cs1.ForeColor = Color.Blue

    c1FlexGrid1.Cols(3).Style = cs1
```

条件付き書式設定

コンテンツに従ってセルの書式設定を行うには、新しいスタイルを作成し、SetCellStyle() メソッドを使用して、特定の条件を満たすセルにスタイルを適用する必要があります。たとえば、指定した値より大きい値を強調表示するには、新しいスタイルを使用して、それらの値を含むセルを書式設定します。



CustomerID	ProductID	PurchaseDate	PaymentAmount	Quantity
12	14	Monday, January 2	$64,000.00	5
32	3	Monday, January 2	$76,800.00	3
15	12	Monday, January 2	*$86,575.00*	5
13	3	Monday, January 2	$51,200.00	2
30	4	Wednesday, Janua	*$118,350.00*	3
15	9	Wednesday, Janua	*$109,800.00*	2
23	8	Thursday, January	$47,780.00	1
12	3	Friday, January 24,	$76,800.00	3
29	8	Friday, January 24,	*$95,560.00*	2
19	10	Saturday, January	*$82,484.00*	2
25	6	Saturday, January	$44,320.00	1
20	15	Saturday, January	$60,000.00	3
14	7	Sunday, January 2	*$148,800.00*	3
22	13	Sunday, January 2	$70,992.00	4

以下のコードは、WinForms FlexGrid のセルに条件付き書式設定を適用する方法を示しています。

C#

```
CellStyle cs;

// 大きな値のカスタムスタイルを作成します
cs = c1FlexGrid1.Styles.Add("LargeValue");
cs.Font = new Font(Font, FontStyle.Italic);
```

```
cs.BackColor = Color.Gold;

for (int row = 1; row < c1FlexGrid1.Rows.Count; row++)
if (Convert.ToDouble(c1FlexGrid1[row, 4]) > 80000)
c1FlexGrid1.SetCellStyle(row, 4, cs);
```

VB.NET

```
Dim cs As CellStyle

    cs = c1FlexGrid1.Styles.Add("LargeValue")
    cs.Font = New Font(Font, FontStyle.Italic)
    cs.BackColor = Color.Gold

    For row As Integer = 1 To c1FlexGrid1.Rows.Count - 1

        If Convert.ToDouble(c1FlexGrid1(row, 4)) > 80000 Then
            c1FlexGrid1.SetCellStyle(row, 4, cs)
        End If
    Next
```

オーナー描画セル

上のセクションでは、グリッドの外観を変更するために、CellStyle オブジェクトを使用して FlexGrid のセルをカスタマイズする方法について説明しています。ただし、グラデーション背景、グラフィックのレンダリングなど、グリッドセルをさらにカスタマイズするには、C1FlexGrid クラスの **DrawMode** プロパティと **OwnerDrawCell** イベントを使用できます。

DrawMode プロパティにより、**OwnerDrawCell** イベントが発生するかどうかが決まります。このイベントを使用して、セルのすべての視覚要素をオーバーライドできます。イベントハンドラで **e.Text** パラメータと **e.Image** パラメータを設定することで、セルに表示されるテキストと画像を変更できます。また、**e.Style** プロパティを設定することで、セルの表示に使用されるスタイルを変更できます。

他のセルに影響しないように、Style パラメータのプロパティは変更しないでください。その代わり、Style パラメータに新しい **CellStyle** オブジェクトを割り当てます。たとえば、e.Style.ForeColor = Color.Red を設定するのではなく、**e.Style = c1FlexGrid1.Styles["RedStyle"]** を使用して、パラメータに完全に新しいスタイルを割り当てます。

独自の描画コードを使用してセルに描画したり、カスタムコードと **e.DrawCell** メソッドの呼び出しを組み合わせることもできます。たとえば、GDI 呼び出しを使用してセルの背景を描画し、次に e.DrawCell を呼び出してセルの境界線とコンテンツを表示することができます。

以下の例で、WinForms FlexGrid は、選択されたセル範囲の背景をグラデーションブラシを使用して描画します。このサンプルコードでは、まず **DrawMode** プロパティに **OwnerDraw** を設定し、次に **LinearGradientBrush** オブジェクトを宣言します。

	EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate
	1	Davolio	Nancy	Sales Representati	Ms.	12/8/1968
	2	Fuller	Andrew	Vice President, Sal	Dr.	2/19/1972
	3	Leverling	Janet	Sales Representati	Ms.	8/30/1983
	4	Peacock	Margaret	Sales Representati	Mrs.	9/19/1957
	5	Buchanan	Steven	Sales Manager	Mr.	3/4/1975
	6	Suyama	Michael	Sales Representati	Mr.	7/2/1983
	7	King	Robert	Sales Representati	Mr.	5/29/1980
	8	Callahan	Laura	Inside Sales Coordi	Ms.	1/9/1978
	9	Dodsworth	Anne	Sales Representati	Ms.	1/27/1986

C#

```
System.Drawing.Drawing2D.LinearGradientBrush m_GradientBrush;
private void Form1_Load(object sender, EventArgs e)
```

```
{
    // オーナー描画セルで使用するブラシ
    m_GradientBrush = new
System.Drawing.Drawing2D.LinearGradientBrush(ClientRectangle, Color.SteelBlue,
Color.White, 45);

    // オーナー描画を使用してグラデーションを追加します
    c1FlexGrid1.DrawMode = C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw;
}

private void C1FlexGrid1_OwnerDrawCell(object sender,
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
{
    // グラデーションブラシを使用して選択したセルの背景を描画します
    if (c1FlexGrid1.Selection.Contains(e.Row, e.Col))
    {
// 背景を描画します
e.Graphics.FillRectangle(m_GradientBrush, e.Bounds);

// グリッドにコンテンツを描画させます
e.DrawCell(C1.Win.C1FlexGrid.DrawCellFlags.Content);

// このセルの描画が完了しました
e.Handled = true;
    }
}
```

VB.NET

```
Private m_GradientBrush As Drawing.Drawing2D.LinearGradientBrush

Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' オーナー描画セルで使用するブラシ
    m_GradientBrush = New Drawing.Drawing2D.LinearGradientBrush(ClientRectangle,
Color.SteelBlue, Color.White, 45)

    ' オーナー描画を使用してグラデーションを追加します
    c1FlexGrid1.DrawMode = C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw
End Sub

Private Sub C1FlexGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs)
    ' グラデーションブラシを使用して選択したセルの背景を描画します
    If c1FlexGrid1.Selection.Contains(e.Row, e.Col) Then
        ' 背景を描画します
        e.Graphics.FillRectangle(m_GradientBrush, e.Bounds)

        ' グリッドにコンテンツを描画させます
        e.DrawCell(C1.Win.C1FlexGrid.DrawCellFlags.Content)

        ' このセルの描画が完了しました
        e.Handled = True
    End If
End Sub
'削除
```

グリッド

このセクションでは、グリッドで実行できるさまざまな操作について説明します。

トピック	スナップショット	コンテンツ
基本的な操作		グリッドで実行できる基本的な操作について説明します。 • グリッドデータの入れ替え • コンテキストメニューの表示
キーボードナビゲーション		次のモードでサポートされているキーとその操作について説明します。 • 非編集モード • 編集モード
パフォーマンスの強化		パフォーマンスを向上させ、グリッドを最大限に活用するためのヒントとテクニックを紹介します。 • データの仮想化を使用する • BeginUpdate メソッドと EndUpdate メソッドを使用する • AutoResize プロパティを False のままにする（デフォルト） • スタイルを動的に割り当てる • OwnerDrawCell イベントでスタイルが変更されないようにする • 列に省略符を表示する • 1 つのセルに複数行のテキストを表示する • データテーブルへの連結時のデータソート順を取得する • 列に文字数の制限を指定する

基本的な操作

このトピックでは、グリッドレベルで処理する必要があるさまざまな操作について説明します。

グリッドデータの入れ替え

データの入れ替えとは、列のデータと行のデータを入れ替えることです。WinForms FlexGrid でこれを行うには、次のコードに示すように、クラスの **Transpose()** メソッドを使用します。

C#

```
c1FlexGrid1.Transpose();
```

VB.NET

```
C1FlexGrid1.Transpose()
```

データの入れ替えは、非連結グリッドの場合にのみ可能です。また、グリッド内の列にソートが適用されている場合は、**Transpose()** メソッドを実行すると、データの入れ替えの前にソートが削除されます。

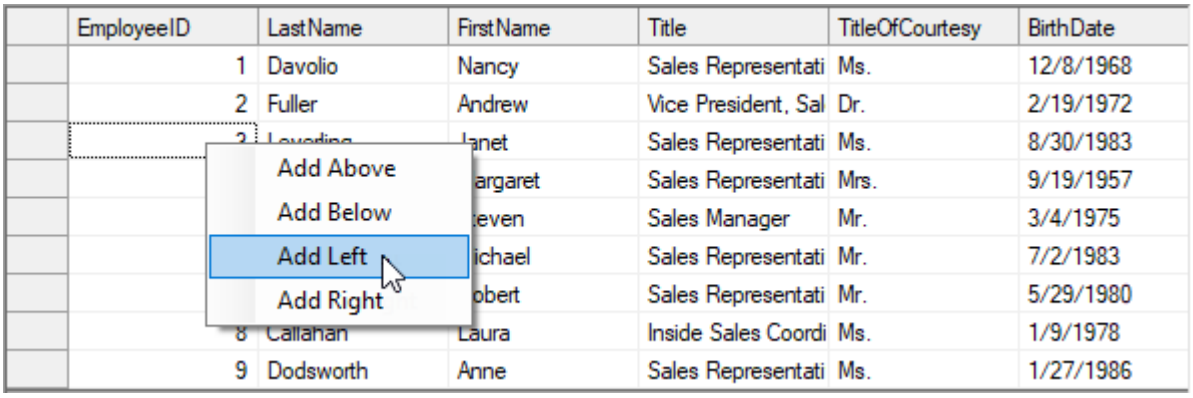
コンテキストメニューの表示

コンテキストメニューには、ユーザーに便利なように、頻繁に使用するアクションのショートカットが表示されます。FlexGrid では、コンテキストメニューが表示される状況が 2 つあります。

- 非編集モードでコンテキストメニューを表示する
- 編集モードでコンテキストメニューを表示する

非編集モードでのコンテキストメニューの表示

グリッドが非編集モードのときにコンテキストメニューを表示するには、ContextMenuStrip コントロールのインスタンスを作成し、メニュー項目を追加し、そのインスタンスを **Control** クラスの ContextMenuStrip プロパティに割り当てる必要があります。



非編集モードで WinForms FlexGrid にコンテキストメニューを表示する方法については、次のコードを参照してください。

C#

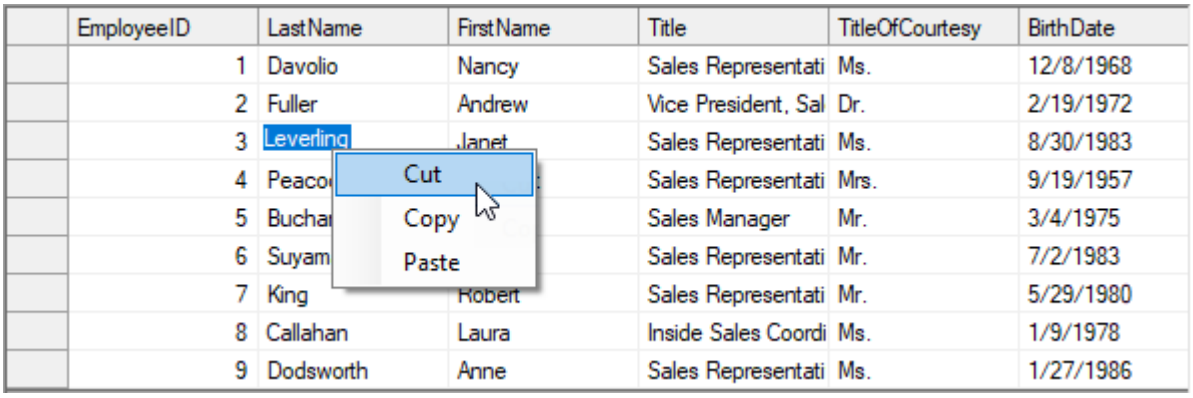
```
// ContextMenuStripコントロールのインスタンスを作成します
ContextMenuStrip cm = new ContextMenuStrip();

// コンテキストメニューにメニュー項目を追加します
cm.Items.Add("Add Above");
cm.Items.Add("Add Below");
cm.Items.Add("Add Left");
cm.Items.Add("Add Right");

// インスタンスをContextMenuStripプロパティに割り当てます
c1FlexGrid1.ContextMenuStrip = cm;
```

VB.NET

```
' ContextMenuStripコントロールのインスタンスを作成します
Dim cm As ContextMenuStrip = New ContextMenuStrip()

' コンテキストメニューにメニュー項目を追加します
cm.Items.Add("Add Above")
cm.Items.Add("Add Below")
cm.Items.Add("Add Left")
cm.Items.Add("Add Right")

' インスタンスをContextMenuStripプロパティに割り当てます
c1FlexGrid1.ContextMenuStrip = cm
```

編集モードでのコンテキストメニューの表示

編集モードでコンテキストメニューを表示するには、クラスの **StartEdit** イベントを使用して、エディタにコンテキストメニューを表示する必要があります。**StartEdit** イベントで、エディタと **ContextMenuStrip** をインスタンス化し、メニュー項目を追加してから、それをエディタの **ContextMenuStrip** プロパティに割り当てます。



編集モードで WinForms FlexGrid にコンテキストメニューを表示するには、次のコードを使用します。

C#

```
private void C1FlexGrid1_StartEdit(object sender, C1.Win.C1FlexGrid.RowColEventArgs
e)
{
     TextBox tb = (TextBox)c1FlexGrid1.Editor;

    // コンテキストメニューを作成します
    ContextMenuStrip cm2 = new ContextMenuStrip();
    cm2.Items.Add("Cut");
    cm2.Items.Add("Copy");
    cm2.Items.Add("Paste");

    // コンテキストメニューを設定します
    tb.ContextMenuStrip = cm2;
}
```

VB.NET

```
Private Sub C1FlexGrid1_StartEdit(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs)
    Dim tb As TextBox = CType(c1FlexGrid1.Editor, TextBox)

    ' コンテキストメニューを作成します
    Dim cm2 As ContextMenuStrip = New ContextMenuStrip()
    cm2.Items.Add("Cut")
    cm2.Items.Add("Copy")
    cm2.Items.Add("Paste")

    ' コンテキストメニューを設定します
    tb.ContextMenuStrip = cm2
End Sub
```

キーボードナビゲーション

FlexGrid では、組み込みのキーボードサポートを使用して、キーを使用するだけでとても簡単にフォーカスの移動、編集モードの開始/終了などの基本的なナビゲーション操作を行うことができます。以下の表に、セルの非編集モードと編集モードでサポートされているキーと、それに対応する操作をまとめます。

非編集モード

キー	キーの動作

↑ ← → ↓	矢印キーが示す方向の隣接するセルにフォーカスを移動します。
Shift + 矢印	矢印キーで示す方向に隣接するセルを複数選択します。
F2	**グリッドが編集可能:** セルを編集モードに切り替えます。セル内に値がある場合は、その値が選択されます。それ以外の場合、グリッドはセル内にカーソルを表示します。 **グリッドが編集不可:** アクションなし。
Enter	**グリッドが編集可能:** セルを編集モードに切り替えます。セル内に値がある場合は、その値が選択されます。それ以外の場合、グリッドはセル内にカーソルを表示します。 **グリッドが編集不可:** 選択を下方向の次の可視行に移動します。 **[Enter]**キーを押したときのデフォルト動作を変更することもできます。それには **KeyActionEnter** プロパティを使用します。
ホーム	行の最初のセルにフォーカスを移動します。
終了	行の最後のセルにフォーカスを移動します。
Ctrl + Home	列の最初のセルにフォーカスを移動します。
Ctrl + End	列の最後のセルにフォーカスを移動します。
Tab	デフォルトで、グリッドは**[Tab]**キーの押下を無視し、フォーム内のコントロールを順番に切り替えます。ただし、**[Tab]**キーを押したときのアクションは、**KeyActionTab** プロパティを使用して定義できます。
Ctrl + C Ctrl + X Ctrl + V	**AutoClipboard** プロパティが **true** に設定されている場合は（デフォルト値は **false**）、通常のクリップボード操作を行います。つまり、Ctrl+C でコピー、Ctrl+X で切り取り、Ctrl+V で貼り付けです。

編集モード

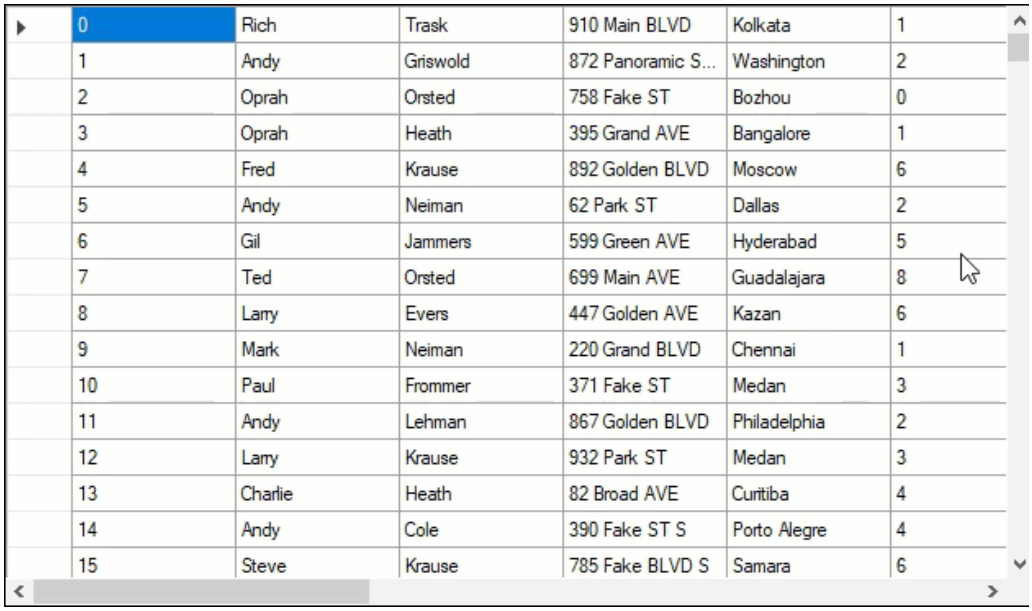
キー	キーの動作
↑ ↓	矢印キーが示す方向の隣接するセルにフォーカスを移動します。

← →	矢印キーで示す方向に 1 文字ずつカーソルを移動します。最初の文字または最後の文字にカーソルがある場合は、矢印キーの方向の隣接するセルにフォーカスを移動します。
Enter	セルを非編集モードに切り替え、フォーカスを下のセルに移動します。[Enter]キーを押したときのデフォルト動作を変更することもできます。それには **KeyActionEnter** プロパティを使用します。
Esc	編集をキャンセルして編集モードを終了します。
Tab	デフォルトで、グリッドは**[Tab]**キーの押下を無視し、フォーム内のコントロールを順番に切り替えます。ただし、[Tab]キーを押したときのアクションは、**KeyActionTab** プロパティを使用して定義できます。

パフォーマンスの強化

データの仮想化を使用する

大規模なデータセットを効率的にレンダリングするために、FlexGrid はデータの仮想化をサポートしており、ユーザーが下にスクロールすると、データはページ単位で取得されます。グリッドは、行の総数を把握していますが、ユーザーに表示される行のみをロードして表示します。たとえば、C1.DataCollection パッケージを使用して、仮想化可能なデータソースを実装できます。C1VirtualDataCollection を継承したクラスを作成し、**GetPageAsync** メソッドを実装します。このメソッドは、割り当てられたデータソースから 1 ページ分のデータを返します。次の GIF は、仮想スクロールモードで FlexGrid がどのように表示されるかを示しています。



0	Rich	Trask	910 Main BLVD	Kolkata	1
1	Andy	Griswold	872 Panoramic S...	Washington	2
2	Oprah	Orsted	758 Fake ST	Bozhou	0
3	Oprah	Heath	395 Grand AVE	Bangalore	1
4	Fred	Krause	892 Golden BLVD	Moscow	6
5	Andy	Neiman	62 Park ST	Dallas	2
6	Gil	Jammers	599 Green AVE	Hyderabad	5
7	Ted	Orsted	699 Main AVE	Guadalajara	8
8	Larry	Evers	447 Golden AVE	Kazan	6
9	Mark	Neiman	220 Grand BLVD	Chennai	1
10	Paul	Frommer	371 Fake ST	Medan	3
11	Andy	Lehman	867 Golden BLVD	Philadelphia	2
12	Larry	Krause	932 Park ST	Medan	3
13	Charlie	Heath	82 Broad AVE	Curitiba	4
14	Andy	Cole	390 Fake ST S	Porto Alegre	4
15	Steve	Krause	785 Fake BLVD S	Samara	6

C#

```
public class VirtualModeCollectionView : C1VirtualDataCollection<Customer>
{
   public int TotalCount { get; set; } = 1_000;
   protected override async Task<Tuple<int, IReadOnlyList<Customer>>> GetPageAsync(int pageIndex, int
startingIndex, int count, IReadOnlyList<SortDescription> sortDescriptions = null, FilterExpression
filterExpression = null, CancellationToken cancellationToken = default(CancellationToken))
   {
      await Task.Delay(500, cancellationToken);//Simulates network traffic.
      return new Tuple<int, IReadOnlyList<Customer>>(TotalCount, Enumerable.Range(startingIndex,
count).Select(i => new Customer(i)).ToList());
   }
}
```

VB.NET

```
Public Class VirtualModeCollectionView
    Inherits C1VirtualDataCollection(Of Customer)

    Public Property TotalCount As Integer = 1_000

    Protected Overrides Async Function GetPageAsync(ByVal pageIndex As Integer, ByVal startingIndex As
Integer, ByVal count As Integer, ByVal Optional sortDescriptions As IReadOnlyList(Of SortDescription) =
Nothing, ByVal Optional filterExpression As FilterExpression = Nothing, ByVal Optional cancellationToken
As CancellationToken = DirectCast(Nothing, CancellationToken)) As Task(Of Tuple(Of Integer,
IReadOnlyList(Of Customer)))
        Await Task.Delay(500, cancellationToken) 'Simulates network traffic.
        Return New Tuple(Of Integer, IReadOnlyList(Of Customer))(TotalCount,
Enumerable.Range(CInt(startingIndex), CInt(count)).[Select](Function(i) New Customer(i)).ToList())
    End Function
End Class
```

BeginUpdate メソッドと EndUpdate メソッドを使用する

BeginUpdate メソッドと **EndUpdate** メソッドを使用して、グリッドのパフォーマンスを最適化できます。大規模な変更を行う前に BeginUpdate を呼び出し、完了したら EndUpdate を呼び出すことで、再描画を保留します. これにより、画面のちらつきが減り、パフォーマンスも向上します。特にグリッドに大量の行を追加する場合は、1 行が追加されるたびに範囲を再計算し、スクロールバーを更新する必要があるため、この最適化が役立ちます。

以下のコードは、大量の行を効率的に WinForms FlexGrid に追加する方法を示します。EndUpdate メソッドを「finally」ブロック内で呼び出すことで、再描画が正しく復元されることに注目してください。

C#

```
void UpdateGrid(C1FlexGrid c1FlexGrid1)
{
  try
  {
    c1FlexGrid1.BeginUpdate(); // ちらつきを避けるためにレンダリングを一時停止します
    c1FlexGrid1.Rows.Count = 1;
    for (int i = 1; i < 10000; i++)
      c1FlexGrid1.AddItem("Row " + i.ToString());
  }
  finally
  {
    c1FlexGrid1.EndUpdate(); // 常にレンダリングを復元します
  }
}
```

VB.NET

```
Private Sub UpdateGrid(ByVal c1FlexGrid1 As C1FlexGrid)
    Try
        c1FlexGrid1.BeginUpdate() ' ちらつきを避けるためにレンダリングを一時停止します
        c1FlexGrid1.Rows.Count = 1

        For i As Integer = 1 To 10000 - 1
            c1FlexGrid1.AddItem("Row " & i.ToString())
        Next

    Finally
        c1FlexGrid1.EndUpdate() ' 常にレンダリングを復元します
    End Try
End Sub
```

AutoResize プロパティを False のままにする（デフォルト）

連結グリッドで AutoResize プロパティが **true** に設定されていると、データソースから新しいデータが読み取られるたびに、列が最大幅のエントリに合わせて自動的にサイズ変更されます。データソースに大量の行や列が含まれている場合は、自動的なサイズ変更に時間がかかる場合があります。このような場合は、**AutoResize** を **false** に設定して、直接コード内で列幅を設定することを検討してください。

スタイルを動的に割り当てる

FlexGrid では、セルのスタイルを作成して、行、列、および任意のセル範囲に割り当てることができます。この機能を使用して、グリッドセルを条件付きで書式設定できます。通常、これは SetCellStyle() メソッドを使用して行います。ただし、その場合は、セルの値が変わるたびにスタイルを更新する必要があります。また、グリッドがデータソースに連結されている場合は、ソートやフィルタ処理などの操作後にデータソースがリセットされるたびにスタイルは失わ

れます。このような場合に優れた方法は、OwnerDraw 機能を使用して、セルの値に基づいて動的にスタイルを選択することです。たとえば、このサンプルコードは、WinForms FlexGrid で負の値を赤色で、1,000 を超える値を緑色で表示する方法を示しています。

C#

```
private void Form1_Load(object sender, EventArgs e)
{
   // 列にランダムな値を入力します
      c1FlexGrid1.Cols[1].DataType = typeof(int);
      Random rnd = new Random();
      for (int r = 1; r < c1FlexGrid1.Rows.Count; r++)
       {
          c1FlexGrid1[r, 1] = rnd.Next(-10000, 10000);
       }

   // 負の値を表示するために使用されるスタイルを作成します
      c1FlexGrid1.Styles.Add("Red").ForeColor = Color.Red;

   // DrawModeプロパティを設定して、OwnerDrawを有効にします
      c1FlexGrid1.DrawMode = C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw;
      c1FlexGrid1.OwnerDrawCell += new
C1.Win.C1FlexGrid.OwnerDrawCellEventHandler(C1FlexGrid1_OwnerDrawCell);
}

private void C1FlexGrid1_OwnerDrawCell(object sender, OwnerDrawCellEventArgs e)
{
    if(!e.Measuring)
    {
        // 行と列に整数データが含まれていることを確認します
        if (e.Row > 0 && c1FlexGrid1.Cols[e.Col].DataType == typeof(int))
        {
           // スタイル「Red」を適用します
           e.Style = c1FlexGrid1.Styles["Red"];
        }
    }
}
```

VB.NET

```
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
        ' 列にランダムな値を入力します
        c1FlexGrid1.Cols(1).DataType = GetType(Integer)
        Dim rnd As Random = New Random()

        For r As Integer = 1 To c1FlexGrid1.Rows.Count - 1
            c1FlexGrid1(r, 1) = rnd.[Next](-10000, 10000)
        Next


        ' 負の値を表示するために使用されるスタイルを作成します
        c1FlexGrid1.Styles.Add("Red").ForeColor = Color.Red

        ' DrawModeプロパティを設定して、OwnerDrawを有効にします
        c1FlexGrid1.DrawMode = C1.Win.C1FlexGrid.DrawModeEnum.OwnerDraw
        c1FlexGrid1.OwnerDrawCell += New C1.Win.C1FlexGrid.OwnerDrawCellEventHandler(AddressOf
C1FlexGrid1_OwnerDrawCell)
    End Sub

    Private Sub C1FlexGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e As OwnerDrawCellEventArgs)
        If Not e.Measuring Then

            ' 行と列に整数データが含まれていることを確認します
            If e.Row > 0 AndAlso c1FlexGrid1.Cols(e.Col).DataType Is GetType(Integer) Then
                ' スタイル「Red」を適用します
                e.Style = c1FlexGrid1.Styles("Red")
            End If
        End If
    End Sub
```

OwnerDrawCell イベントでスタイルが変更されないようにする

パフォーマンスを改善する別の方法としては、**OwnerDrawCell** イベントのパラメータとして渡される CellStyle オブジェクトを変更しないことがあります。代わりに、**e.Style** パラメータに新しい値を割り当てます。イベントハンドラに渡される **CellStyle** は別のセルからもよく使用されるため、これは重要です。たと

えば、意図せず WinForms FlexGrid の標準スタイルを変更してしまうと、グリッドの他の類似のセルにも影響を与えます。

C#

```
// ** 正しい方法：
private void C1FlexGrid1_OwnerDrawCell(object sender, C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
{
    // このセルをペイントするときに使用するスタイルを選択します
    e.Style = MyStyleSelector(e.Row, e.Col);
}

// ** 正しくない方法：
private void C1FlexGrid1_OwnerDrawCell(object sender, C1.Win.C1FlexGrid.OwnerDrawCellEventArgs e)
{
    // このセルをペイントするときに使用するスタイルを選択します
    // CellStyleオブジェクトを変更するとグリッドが無効になり、
    // このイベントハンドラーが何度も呼び出されるため、
    // このメソッドは正しくありません。
    e.Style.Color = MyColorSelector(e.Row, e.Col);
}
```

VB.NET

```
' ** 正しい方法：
Private Sub C1FlexGrid1_OwnerDrawCellMethod(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs)
' このセルをペイントするときに使用するスタイルを選択します
e.Style = MyStyleSelector(e.Row, e.Col)
End Sub


' ** 正しくない方法：
Private Sub C1FlexGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.OwnerDrawCellEventArgs)
' このセルをペイントするときに使用するスタイルを選択します
' CellStyleオブジェクトを変更するとグリッドが無効になり、
' このイベントハンドラーが何度も呼び出されるため、
' このメソッドは正しくありません。
e.Style.Color = MyColorSelector(e.Row, e.Col)
End Sub
```

列に省略符を表示する

グリッドの 1 つの列に省略記号を表示するには、Trimming プロパティを使用する必要があります。セルに収まるように文字列をトリミングする長さを決定するには、Trimming プロパティを **None**、**Character**、**Word**、**EllipsisCharacter**、**EllipsisWord**、または **EllipsisPath** に設定します。トリミングの詳細については、「トリミングされたテキストの表示」を参照してください。

次のコードは、一番近い文字までテキストをトリミングして、WinForms Flexgrid の 2 番目の列の最後に省略記号を表示するように **Trimming** プロパティを設定します。

C#

```
c1FlexGrid1.Cols[1].StyleNew.Trimming =StringTrimming.EllipsisCharacter;
```

VB.NET

```
c1FlexGrid1.Cols(1).StyleNew.Trimming = StringTrimming.EllipsisCharacter
```

1 つのセルに複数行のテキストを表示する

1 つのセル内に複数のテキスト行を表示するには、WordWrap プロパティと **Height** プロパティを使用します。**WordWrap** プロパティは、スペースが含まれる長い文字列を自動的に改行して複数行に表示するどうかを決定します。強制改行（vbCrLf または "\n\r"）が含まれる文字列は常に複数行に表示されます。複数行テキストは、固定セルにもスクロール可能なセルにも表示できます。テキストの折り返しについては、「テキストの折り返し」を参照してください。

WinForms FlexGrid で複数行テキストを効率的に表示する方法については、以下のコードを参照してください。

C#

```
// WordWrapプロパティを設定します
c1FlexGrid1.Styles["Normal"].WordWrap = true;

// 行の高さを設定します
c1FlexGrid1.Rows[1].Height = 2 * c1FlexGrid1.Rows.DefaultSize;

// セルにテキストを追加します
c1FlexGrid1[1, 2] = "This is the first line. \r\n This is the second line.";
```

VB.NET

```
    ' WordWrapプロパティを設定します
    c1FlexGrid1.Styles("Normal").WordWrap = True

    ' 行の高さを設定します
    c1FlexGrid1.Rows(1).Height = 2 * c1FlexGrid1.Rows.DefaultSize

    ' セルにテキストを追加します
    c1FlexGrid1(1, 2) = "This is the first line. " & vbCrLf & " This is the second line."
```

データテーブルへの連結時のデータソート順を取得する

データがリフレッシュされたときにグリッドのソート方法を保持するには、デフォルトビューの Sort プロパティとソート式を使用します。Sort プロパティは、列名の後に ASC（デフォルト）または DESC を付けた文字列を使用して、昇順または降順で列をソートします。複数の列は、列名をカンマで区切って指定することでソートできます。1 つのソート式には、複数のグリッド列の名前または 1 つの計算値を含めることができます。実行時にソート式を設定すると、変更がデータビューに即座に反映されます。

次のコードは、WinForms FlexGrid の Sort プロパティでソート式を使用する方法を示しています。

C#

```
// UnitsInStock列、次にProductID列でデータを並べ替えます
   this.productsBindingSource.Sort = "UnitsInStock ASC, ProductID ASC";
```

VB.NET

```
' UnitsInStock列、次にProductID列でデータを並べ替えます
Me.productsBindingSource.Sort = "UnitsInStock ASC, ProductID ASC"
```

列に文字数の制限を指定する

任意の列にユーザーが入力できる最大の文字数を設定するには、**SetupEditor** イベントを使用します。C1FlexGrid クラスの **StartEdit** イベントで **C1TextBox** などの外部エディタを宣言する必要があります。次に、**SetupEditor** イベントで、1 つの列セルに格納できる最大文字数を設定できます。

WinForms FlexGrid 列に文字数制限を設定するには、次のコードを使用します。

C#

```
private void C1FlexGrid1_StartEdit(object sender, C1.Win.C1FlexGrid.RowColEventArgs e)
{
    c1FlexGrid1.Editor = c1TextBox;
}

private void C1FlexGrid1_SetupEditor(object sender, RowColEventArgs e)
{

   // 3番目の列を20文字に設定し、残りは10文字のみにします
   if (e.Col == 2)
     c1TextBox.MaxLength = 20;
   else
     c1TextBox.MaxLength = 10;
}
```

VB.NET

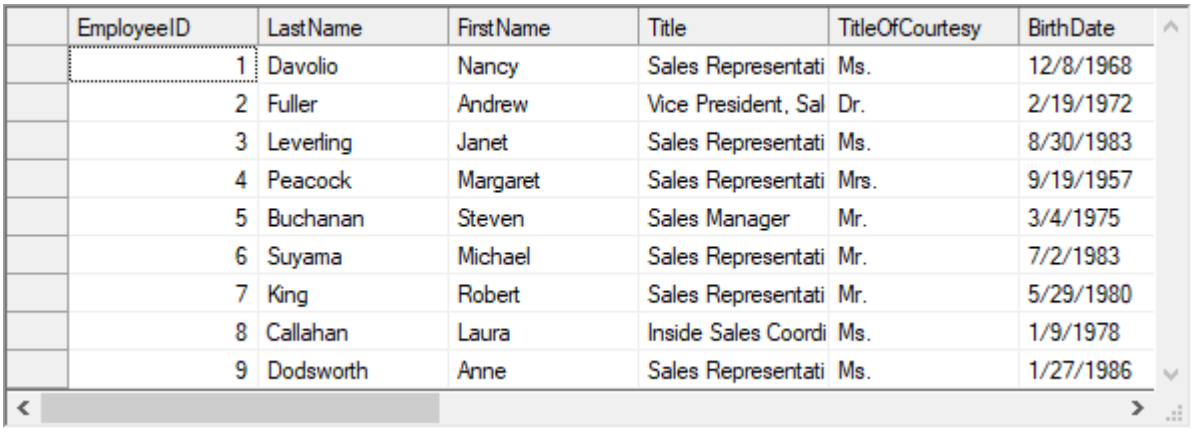
```
    Private Sub C1FlexGrid1_StartEdit(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs)
```

```
    c1FlexGrid1.Editor = c1TextBox
End Sub

Private Sub C1FlexGrid1_SetupEditor(ByVal sender As Object, ByVal e As RowColEventArgs)

    ' 3番目の列を20文字に設定し、残りは10文字のみにします
    If e.Col = 2 Then
        c1TextBox.MaxLength = 20
    Else
        c1TextBox.MaxLength = 10
    End If
End Sub
```

スクロールバー

スクロールバーの表示/非表示

FlexGrid では、ScrollBars プロパティを使用してスクロールバーの表示を管理できます。このプロパティに **ScrollBars** 列挙の値を指定して、スクロールバーを水平方向、垂直方向、またはその両方に表示するか、スクロールバーを非表示にするかを選択します。



	EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate
	1	Davolio	Nancy	Sales Representati	Ms.	12/8/1968
	2	Fuller	Andrew	Vice President, Sal	Dr.	2/19/1972
	3	Leverling	Janet	Sales Representati	Ms.	8/30/1983
	4	Peacock	Margaret	Sales Representati	Mrs.	9/19/1957
	5	Buchanan	Steven	Sales Manager	Mr.	3/4/1975
	6	Suyama	Michael	Sales Representati	Mr.	7/2/1983
	7	King	Robert	Sales Representati	Mr.	5/29/1980
	8	Callahan	Laura	Inside Sales Coordi	Ms.	1/9/1978
	9	Dodsworth	Anne	Sales Representati	Ms.	1/27/1986

次のコードは、WinForms FlexGrid に垂直方向と水平方向のスクロールバーを常に表示する方法を示しています。

C#

```
// 水平および垂直スクロールバーを表示します
c1FlexGrid1.ScrollBars = ScrollBars.Both;

// 常にスクロールバーを表示します
c1FlexGrid1.ScrollOptions = ScrollFlags.AlwaysVisible;
```

VB.NET

```
' 水平および垂直スクロールバーを表示します
c1FlexGrid1.ScrollBars = ScrollBars.Both

' 常にスクロールバーを表示します
c1FlexGrid1.ScrollOptions = ScrollFlags.AlwaysVisible
```

スクロール位置の設定

FlexGrid を特定の位置にスクロールするには、C1FlexGrid クラスの **TopRow** プロパティと **LeftCol** プロパティを設定します。**TopRow** プロパティはグリッドを垂直方向にスクロールし、**LeftCol** プロパティはグリッドの水平方向のスクロール位置を設定します。これらのプロパティの最大値は、行や列の合計数と、グリッドに表示可能な行や列の数によって異なります。この機能は、複数のグリッドを同期してスクロールする場合などに特に便利です。

WinForms FlexGrid のスクロール位置を設定するには、次のコードを使用します。

C#

```
// スクロール位置を設定します
c1FlexGrid1.TopRow = 3;
c1FlexGrid1.LeftCol = 2;
```

VB.NET

```
' スクロール位置を設定します
```

```
c1FlexGrid1.TopRow = 3
c1FlexGrid1.LeftCol = 2
```

その他のスクロールオプション

FlexGrid では、ScrollOptions プロパティを使用して、スクロールバーの表示をさらに細かく制御することもできます。このプロパティは、スクロールバーのオプションをカスタマイズするためのScrollFlags 列挙の値を受け取ります（次の表を参照）。

値	スクロール操作
AlwaysVisible	スクロールバーが無効になっている場合、またはスクロール可能な領域がない場合でも、スクロールバーを表示します。
DelayedScroll	ユーザーがスクロールボックスを放した後にのみ、コンテンツをスクロールします。
KeepMergedRangePosition	結合範囲の最初のセルにスクロール位置を設定できません。
None	デフォルトのスクロール動作を使用します。
ScrollByRowColumn	コンテンツをピクセル単位でスクロールするのではなく、行または列の単位でスクロールします。
ShowScrollTips	垂直方向のスクロールバーをスクロールする際に、**ShowScrollTip** イベントを発生させ、そのバーの横にツールチップを表示します。

次のコードは、WinForms FlexGrid を行または列単位でのみスクロールする方法を示しています。

C#

```
// 行や列単位でスクロールします
c1FlexGrid1.ScrollOptions = ScrollFlags.ScrollByRowColumn;
```

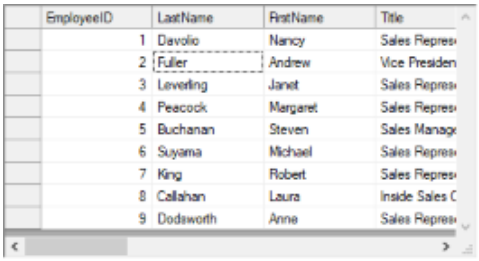
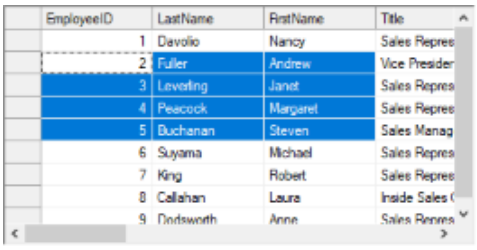
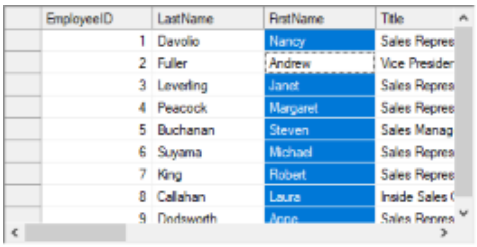
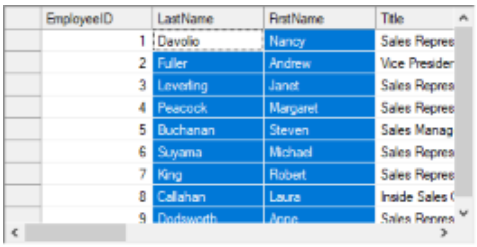
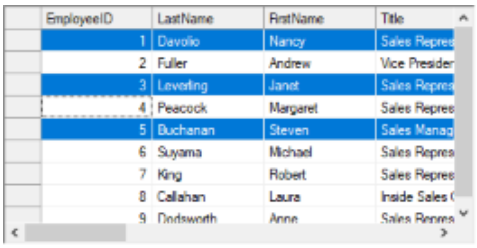
VB.NET

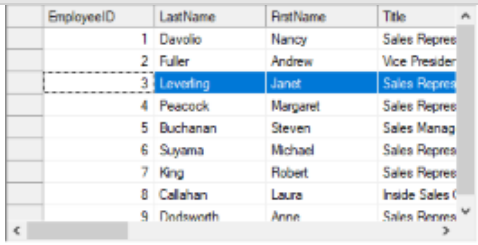
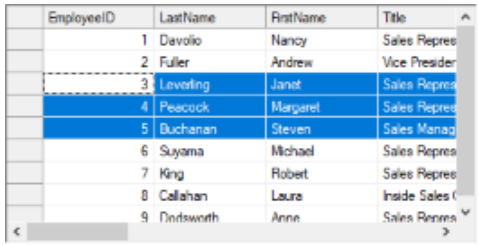
```
' 行や列単位でスクロールします
c1FlexGrid1.ScrollOptions = ScrollFlags.ScrollByRowColumn
```

選択範囲

選択モード

FlexGrid では、デフォルトで、マウスまたはキーボードを使用して隣接する複数のセルを一度に選択したり、行または列のヘッダーをクリックしてその行または列全体を選択することができます。ただし、このデフォルトの動作を変更して、セル、行、列などの単位で選択することもできます。それには、C1FlexGrid クラスの **SelectionMode** プロパティを使用します。このプロパティは、SelectionModeEnum 列挙に含まれる値を受け取ります。 次の表内のスナップショットは、以下の各モードで要素がどのように選択されるかを示しています。

値	説明	スナップショット
Default	マウスまたはキーボードを使用して、隣接する複数のセルを一度に選択できます。また、行または列のヘッダーをクリックすることで、その行または列全体を選択できます。	
Cell	一度に 1 個のセルを選択できます。	
CellRange	マウスまたはキーボードを使用して、隣接する複数のセルを一度に選択できます。	
Column	一度に 1 個の列を選択できます。	
ColumnRange	一度に隣接する複数の列を選択できます。	
ListBox	**[Ctrl]**キーを使用して、隣接していない複数の行を選択できます。	

値	説明	スナップショット
Row	一度に 1 つの行を選択できます。	
RowRange	一度に隣接する複数の行を選択できます。	

選択範囲の設定

FlexGrid では、コードを使用して、さまざまな方法で選択範囲を設定できます。1 個のセル、セル範囲、複数の行など、選択範囲の要件に応じて、以下の方法を使用できます。

選択範囲	メソッド/プロパティ	サンプルコード

1 個のセル	**Row** プロパティと **Col** プロパティを設定します。これらのプロパティのデフォルト値は **1** です。したがって、デフォルトでは、グリッドの左上にある最初のスクロール可能セルに選択範囲が設定されます。	Example Title
		`c1FlexGrid1.Row = 2;`
		`c1FlexGrid1.Col = 1;`

	1 個のセルを選択するには、**Select**(rowIndex, colIndex) メソッドを呼び出します。	Example Title `c1FlexGrid1.Select(2, 1);`

セル範囲	**RowSel** プロパティと **ColSel** プロパティを設定します。選択範囲は、**Row** プロパティと **Col** プロパティで設定された値から、指定された行と列までに設定されます。ブロック選択範囲を指定するには、**RowSel** と **ColSel** を設定する前に **Row** と **Col** を設定する必要があります。	Example Title
		`c1FlexGrid1.Row = 2;`
		`c1FlexGrid1.Col = 1;`
		`c1FlexGrid1.RowSel = 4;`
		`c1FlexGrid1.ColSel = 3;`

	1 回の呼び出しで 1 つのセル範囲を選択するには、**Select**(CellRange, Boolean) メソッドを呼び出します。	Example Title
		`CellRange cellRange = new CellRange();`
		`cellRange.r1 = 2;`
		`cellRange.r2 = 4;`
		`cellRange.c1 = 1;`
		`cellRange.c2 = 3;`
		`c1FlexGrid1.Select(cellRange);`

行（SelectionModes = SelectionModesEnum.ListBox の場合）	隣接していない行を選択するには、それぞれの行オブジェクトに対して、**Row.Selected** プロパティを **true** に設定します。	Example Title
		`c1FlexGrid1.SelectionMode = SelectionModeEnum.ListBox;`
		`c1FlexGrid1.Rows[1].Selected = true;`
		`c1FlexGrid1.Rows[4].Selected =`

選択範囲	メソッド/プロパティ	サンプルコード
		true;

選択範囲の取得

WinForms FlexGrid の選択範囲を取得するには、C1FlexGrid クラスの **Selection** プロパティを使用します。

C#

```
 private void Button1_Click(object sender, EventArgs e)
   {
     C1.Win.C1FlexGrid.CellRange cr;
     cr = c1FlexGrid1.Selection;
     MessageBox.Show("Selected range\n" +
     cr.r1 + ":" + cr.c1 + " to " + cr.r2 + ":" + cr.c2);
    }
```

VB.NET

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim cr As C1.Win.C1FlexGrid.CellRange
    cr = C1FlexGrid1.Selection
    MessageBox.Show("Selected range" & vbLf & cr.r1.ToString() & ":" + cr.c1.ToString() & " to
" + cr.r2.ToString() & ":" + cr.c2.ToString())
End Sub
```

ダブルクリックでのテキスト選択

デフォルトでは、グリッドセルをダブルクリックすると、セルの状態が編集モードに変わり、マウスポインタの位置にカーソルが表示されます。この動作を変更して、セルのダブルクリックでセルの値を選択することができます。それには、**BeforeDoubleClick** イベントでダブルクリックを検出して無効にします。次に、**StartEditing** メソッドを呼び出して編集モードに入り、エディタを **TextBox** に変更してから、**SelectAll** メソッドを呼び出してセル値を選択します。

次のコードは、ダブルクリックで WinForms FlexGrid のセルのテキストを選択する方法を示しています。

C#

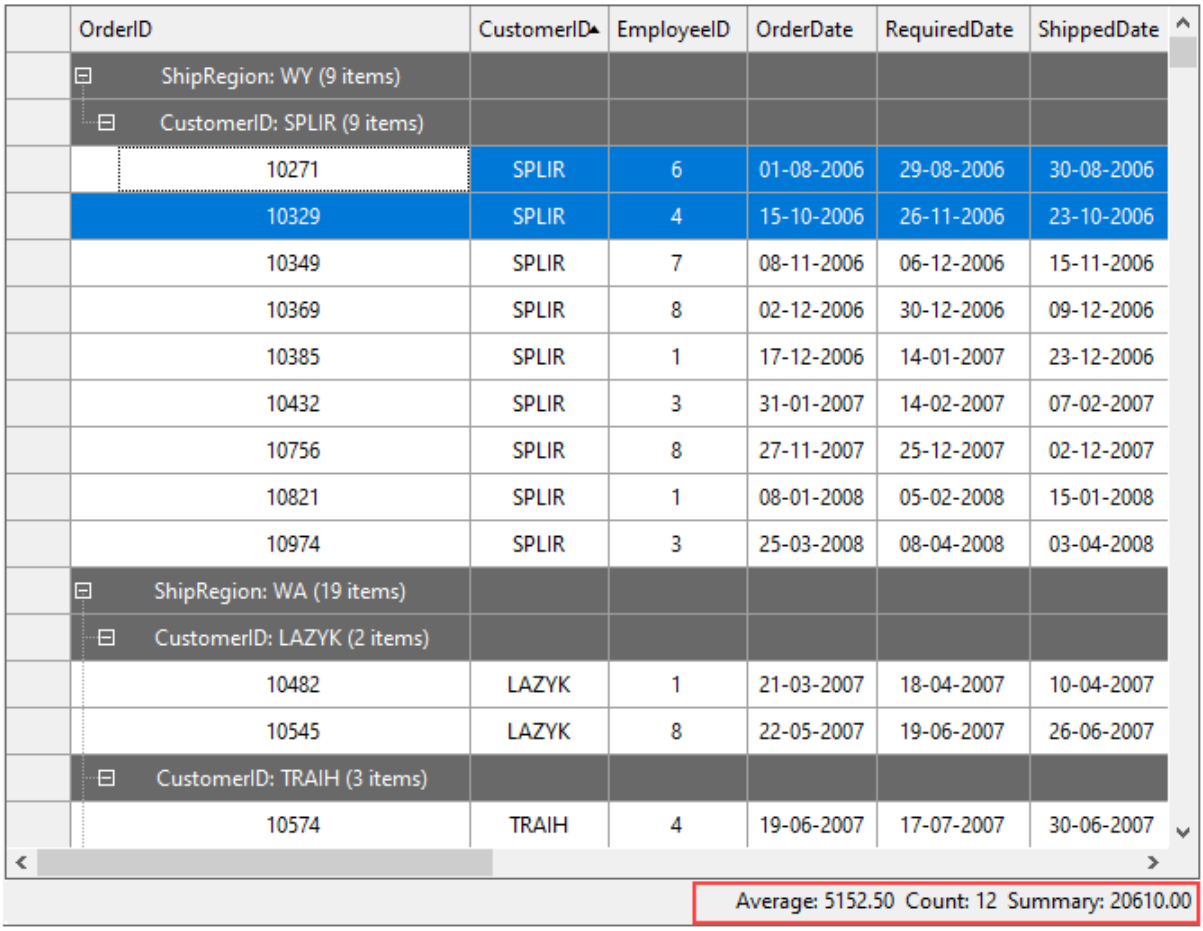
```
private void C1FlexGrid1_BeforeDoubleClick(object sender, BeforeMouseDownEventArgs e)
  {
    // デフォルトのダブルクリックを無効にします
    e.Cancel = true;
    // 編集モードに入ります
    c1FlexGrid1.StartEditing();
    // TextBoxクラスに変換し、SelectAllメソッドを使用して選択します
    TextBox tb = (TextBox)c1FlexGrid1.Editor;
    tb.SelectAll();
  }
```

VB.NET

```
Private Sub C1FlexGrid1_BeforeDoubleClick(ByVal sender As Object, ByVal e As
BeforeMouseDownEventArgs)
    ' デフォルトのダブルクリックを無効にします
    e.Cancel = True
    ' 編集モードに入ります
    c1FlexGrid1.StartEditing()
    ' TextBoxクラスに変換し、SelectAllメソッドを使用して選択します
    Dim tb As TextBox = CType(c1FlexGrid1.Editor, TextBox)
    tb.SelectAll()
End Sub
```

Show Selection Statistics

FlexGrid provides selection statistics, which can be used to display Excel-style data summaries in the footer. It allows you to display the statistics of the basic functions and operations performed on the selected cell range. These functions or operations include aggregate operations such as sum, average, count, maximum number, minimum number, and count distinct which can be applied to the selected cells from the context menu. You can use the **Aggregate** property of the **AggregateDefinition** class to provide data summary for the selected cell range.



The **Aggregate** property uses **AggregateEnum** enumeration to set the aggregate function to be applied on the cells by setting one of the following values:

- **Average**: Displays the value of the non-empty cells in a range.
- **Sum**: Displays the sum of all values in the range.
- **Clear**: Clear the existing aggregates
- **Count**: Displays the total number of non-empty cells in a range.
- **CountDistinct**: Displays the count of unique non-empty cells in a range.
- **Min**: Displays the minimum value in a range.
- **Max**: Displays the maximum value in a range.
- **Percent**: Displays the percentage value of the grand total.
- **Std**: Displays the sample standard deviation of the values in a range.
- **Var**: Displays the sample variance of the values.
- **StdPop**: Displays the population standard deviation of the values in a range (uses the formula based on n).
- **VarPop**: Displays the population variance of the values in a range (uses the formula based on n).
- **Aggregate**: No aggregate.

To show the selection statistics at the footer of the grid, we have added a ToolStripLabel named 'tslSelectionStatistics' docked at the bottom of the grid. Then, add the below code to the **SelChange** event of the **C1FlexGridBase** class. This event fires when the user extends the selection with the mouse in the grid.

C#

```
private void c1FlexGrid1_SelChange(object sender, EventArgs e)
{
```

```
    var text = string.Empty;
    if (!flexGrid1.Selection.IsSingleCell)
    {
        text = $"Average: {flexGrid1.Aggregate(AggregateEnum.Average):F2}  " +
       $"Count: {flexGrid1.Aggregate(AggregateEnum.Count)}  " +
       $"Summary: {flexGrid1.Aggregate(AggregateEnum.Sum):F2}";
    }
    //Gets the text to be displayed on the toolstrip label
    tslSelectionStatistics.Text = text;
}
```

編集

このトピックでは、編集に関連するさまざまなイベントやメソッド、および編集を無効化する方法について説明します。

トピック	コンテンツ
編集モード	編集に関連するさまざまなイベントおよびメソッドについて説明します。
編集の無効化	グリッドでの編集をさまざまなレベルで無効化する方法について説明します。 • グリッド編集の無効化 • 行/列編集の無効化 • セル編集の無効化

編集モード

FlexGrid では、マウスクリックまたはキーボードを使用して、実行時に編集モードを起動できます。グリッドが編集モードかどうかをプログラムによって判定するには、**Editor** プロパティの値を読み取ります。グリッドが編集モードの場合、このプロパティは、エディタとして使用されている **TextBox**、**ComboBox** などのコントロールへの参照を返します。一方、グリッドが編集モードでない場合、このプロパティは **null** を返します。

プログラムによってグリッドを編集モードにするには、StartEditing メソッドを使用し、編集を終了するには、FinishEditing メソッドを呼び出します。また、PreserveEditMode プロパティを使用すると、セル間を移動している間も編集モードの状態を維持できます。

さらに、FlexGrid は、さまざまなイベントを発生させることで、編集プロセスに対して適切な制御を簡単に行えるようにしています。編集プロセス中にグリッドから発生する一連のイベントを次の表にリストします。

イベント名	説明
BeforeEdit	このイベントは、編集可能なセルが選択されるたびに発生します。このイベントの **Cancel** パラメータを **true** に設定することで、セルの編集を禁止することができます。また、**ComboList** プロパティを変更して、適切なドロップダウンボタンをセル内に描画することもできます。ただし、このイベントの後にユーザーが実際に編集を開始せず、単に別のセルまたはコントロールに選択が移動することもあります。
StartEdit	このイベントは **BeforeEdit** に似ていますが、ユーザーが実際にキー入力を行うかセル内のドロップダウンボタンをクリックして、実際に編集を開始しようとしている点が異なります。この段階では、まだ編集を取り消すことができます。この時点では、コントロールが使用するエディタタイプがまだ決まらないため、**Editor** プロパティが **null** であることに注意してください。この段階で、Editor プロパティにカスタムエディタを割り当てることができます。
ChangeEdit	このイベントは、**editor.TextChanged** イベントのラッパーです。このイベントは、エディタの内容が変更されたときに発生します。このイベントを使用して、エディタの現在の内容を追跡することができます。
SetupEditor	このイベントは、エディタコントロールが作成され、セルを編集するように構成された後で、表示される前に発生します。この段階で、エディタのプロパティを変更できます（たとえば、TextBox エディタで使用する最大長、パスワード文字などを設定できます）。独自のイベントハンドラをエディタにアタッチすることもできます。
ValidateEdit	このイベントは、ユーザーが編集を完了したときに、エディタの値がグリッドにコピーされる前に発生します。グリッドから元の値を取得して、その値を調べることができます（このイベントは、セルの座標を提供します）。**Editor** のプロパティ（Editor.Text など）を使用して、グリッドに割り当てられようとしている新しい値を調べることができます。新しい値がセルに対して有効でない場合は、**Cancel** パラメータを **true** に設定すると、グリッドは編集モードのままになります。セルを編集モードのままにするのではなく、元の値を復元して編集モードを終了する場合は、**Cancel** パラメータを **true** に設定し、さらに **FinishEditing** メソッドを呼び出します。
LeaveEdit	このイベントは、グリッドコントロールが編集モードを終了した後に発生します。このイベントを使用して、新しいセルコンテンツを承認または拒否したり、コミットされようとしているエディタの内容を変更できます。

イベント名	説明
AfterEdit	このイベントは、新しい値がセルに適用され、エディタが非アクティブ化された後に発生します。このイベントを使用して、セルの値に依存しているすべての項目（小計、ソートなど）を更新できます。

また、キーボード操作に関連付けられ、キーが押されると発生するイベントがいくつかあります。これらのイベントは、グリッドが編集モードのときに発生すること以外は、System.Windows.Forms.Control クラスの対応するイベントと同じです。

イベント	説明
KeyDownEdit	このイベントは、グリッドが編集モードのときに、キーが押されると発生します。このイベントを使用すると、アクションを 1 回実行したり、カーソルを移動する場合など、キーが押されたままのときには複数回のアクションを実行することができます。
KeyPressEdit	このイベントは、グリッドが編集モードのときに、文字キーが押されると発生します。このイベントを使用して、キー入力に関連する操作（セルエディタ内の入力の処理など）を実行できます。
KeyUpEdit	このイベントは、グリッドが編集モードのときに、キーが放されると発生します。このイベントを使用して、KeypressEdit ロジックが適用された後に実行するロジックを置くことができます。

編集の無効化

FlexGrid は、デフォルトでは、実行時のセル値の編集をエンドユーザーに許可します。ただし、FlexGrid が提供するさまざまなプロパティを使用して、エンドユーザーがどの程度編集を制御できるかを簡単に管理できます。

グリッド編集の無効化

WinForms FlexGrid 全体の編集を無効にするには、次のコードに示すように、C1FlexGrid クラスの AllowEditing プロパティを **false** に設定する必要があります。

C#

```
// グリッドで編集を無効にします
c1FlexGrid1.AllowEditing = false;
```

VB.NET

```
' グリッドで編集を無効にします
c1FlexGrid1.AllowEditing = False
```

行/列編集の無効化

WinForms FlexGrid の特定の行/列の編集を無効にするには、次のコードに示すように Row または Column オブジェクトの **AllowEditing** プロパティを **false** に設定します。

C#

```
// 3行目の編集を無効にします
c1FlexGrid1.Rows[3].AllowEditing = false;

// 3列目の編集を無効にします
c1FlexGrid1.Cols[3].AllowEditing = false;
```

VB.NET

```
' 3行目の編集を無効にします
```

```
c1FlexGrid1.Rows(3).AllowEditing = False

' 3列目の編集を無効にします
c1FlexGrid1.Cols(3).AllowEditing = False
```

セル編集の無効化

特定のセルの編集を無効にするには、**BeforeEdit** イベントを使用し、特定のセルの **Cancel** パラメータを **true** に設定します。

C#

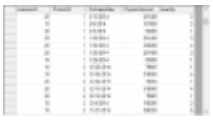
```
// セル編集を無効にします
private void C1FlexGrid1_BeforeEdit(object sender, RowColEventArgs e)
    {
      if ((e.Col == 4) && (e.Row == 2))
       {
           e.Cancel = true;
       }
    }
```

VB.NET

```
' セル編集を無効にします
Private Sub C1FlexGrid1_BeforeEdit(ByVal sender As Object, ByVal e As
RowColEventArgs)
    If e.Col = 4 AndAlso e.Row = 2 Then
        e.Cancel = True
    End If
End Sub
```

ソート

ソートは、グリッドに必要な基本機能の 1 つです。FlexGrid では、昇順または降順でデータをどのようにソートするかを完全に制御できるので、データを簡単に分析できます。このトピックでは、FlexGrid でのソートに関連する操作について説明します。

トピック	スナップショット	コンテンツ
ソートの操作		ソートに関するさまざまな操作について説明します。 • コードによるソート • 複数の列のソート • ソートを元に戻す/やり直す • 特定の列のソートを無効にする • ソート順 • カスタムソート
ソートインジケータ		ソートインジケータを非表示にする方法、配置する方法、およびカスタマイズする方法について説明します。 • ソートインジケータの表示/非表示 • ソートインジケータの配置 • ソートインジケータのカスタマイズ

ソートの操作

デフォルトでは、FlexGrid のエンドユーザーは、列ヘッダーをクリックすることで、1 つの列に昇順または降順のソートを適用できます。ただし、必要に応じてコードでデータのソートを実行できる柔軟性もあります。以下のセクションでは、ソートに関するさまざまな操作を実行する方法について説明します。

CustomerID	ProductID	PurchaseDate	PaymentAmount	Quantity
25	1	2/13/2014	251400	3
15	1	2/6/2014	167600	2
20	1	2/2/2014	83800	1
29	1	1/30/2014	251400	3
20	1	1/30/2014	335200	4
29	1	1/29/2014	251400	3
14	1	1/26/2014	83800	1
12	2	12/20/2014	79645	1
19	2	12/20/2014	318580	4
26	2	12/20/2014	79645	1
28	2	12/15/2014	318580	4
30	2	12/14/2014	79645	1
18	2	12/6/2014	159290	2
19	2	11/27/2014	398225	5

コードによるソート

コードでソートを適用するには、C1FlexGrid クラスの Sort メソッドを呼び出します。このメソッドは、SortFlags 列挙をパラメータとして受け取ります。この列挙により、ソート順の設定、大文字小文字を区別しないなど、さまざまなソートオプションを提供できます。さまざまなオーバーロードメソッドを利用することで、1 つの列、セル範囲、行範囲、または列範囲に柔軟にソートを適用できます。

WinForms FlexGrid においてコードで列をソートし、ソートオプションを適用するには、次のコードを使用します。

C#

```
//方法 1:
//2番目の列を降順でソートします
c1FlexGrid1.Sort(C1.Win.C1FlexGrid.SortFlags.Descending, 2);

//方法 2:
//SortFlagを使用して複数のソートオプションを指定します
C1.Win.C1FlexGrid.SortFlags order = C1.Win.C1FlexGrid.SortFlags.Ascending |
C1.Win.C1FlexGrid.SortFlags.IgnoreCase;

//Sortメソッドを呼び出します
c1FlexGrid1.Sort(order, 2);
```

VB.NET

```
' 方法 1:
' 2番目の列を降順でソートします
c1FlexGrid1.Sort(C1.Win.C1FlexGrid.SortFlags.Descending, 2)

' 方法 2:
' SortFlagを使用して複数のソートオプションを指定します
Dim order As C1.Win.C1FlexGrid.SortFlags = C1.Win.C1FlexGrid.SortFlags.Ascending Or
C1.Win.C1FlexGrid.SortFlags.IgnoreCase

' Sortメソッドを呼び出します
c1FlexGrid1.Sort(order, 2)
```

複数の列のソート

コードを使用して、複数の列にソートを適用するには、Column クラスの **Sort** プロパティを使用し、**SortFlags** を **UseColSort** に設定して **Sort()** メソッドを呼び出します。**Sort** プロパティは、**SortFlags** 列挙に含まれる値を受け取ります。

次のコードは、コードを使用して WinForms FlexGrid の複数の列をソートする方法を示しています。

C#

```
//複数の列にソートを適用します
c1FlexGrid1.Cols[2].Sort = SortFlags.Ascending;
c1FlexGrid1.Cols[3].Sort = SortFlags.Descending;

//Sortメソッドを呼び出します
c1FlexGrid1.Sort(SortFlags.UseColSort, 2, 3);
```

VB.NET

```
'複数の列にソートを適用します
c1FlexGrid1.Cols(2).Sort = SortFlags.Ascending
c1FlexGrid1.Cols(3).Sort = SortFlags.Descending

'Sortメソッドを呼び出します
c1FlexGrid1.Sort(SortFlags.UseColSort, 2, 3)
```

ユーザーが実行時に複数の列をソートできるようにするには、C1FlexGrid クラスの AllowSorting プロパティを **MultiColumn** に設定します。このプロパティは、AllowSortingEnum 列挙に含まれる値を受け取ります。

次のコードは、実行時にユーザーが WinForms FlexGrid の複数の列をソートできるようにする方法を示しています。

C#

```
// グリッドの複数の列でのソートを許可します
c1FlexGrid1.AllowSorting = AllowSortingEnum.MultiColumn;
```

VB.NET

```
' グリッドの複数の列でのソートを許可します
c1FlexGrid1.AllowSorting = AllowSortingEnum.MultiColumn
```

ソートを元に戻す/やり直す

グリッドからソートを削除するには、**C1FlexGrid** クラスの **SortDefinition** プロパティを空の文字列に設定します。

次のコードは、WinForms FlexGrid からソートを削除する方法を示しています。

C#

```
// ソートを削除します
c1FlexGrid1.SortDefinition = string.Empty;
```

VB.NET

```
' ソートを削除します
c1FlexGrid1.SortDefinition = String.Empty
```

特定の列のソートを無効にする

特定の列のソートを無効にするには、その Column オブジェクトの **AllowSorting** プロパティを **false** に設定する必要があります。

WinForms FlexGrid の特定の列のソートを無効にするには、次のコードを使用します。

C#

```
// 特定の列のソートを無効にします
c1FlexGrid1.Cols[2].AllowSorting = false;
```

VB.NET

```
' 特定の列のソートを無効にします
c1FlexGrid1.Cols(2).AllowSorting = False
```

ソート順

通常、ソートの順序は、連結モードと非連結モードで異なります。連結モードの場合は、列ヘッダーがクリックされると、データテーブルの **DefaultView.Sort** プロパティと同様にソートが行われます。非連結モードの場合は、String.Compare メソッドに従って列がソートされるか、カルチャによっては大文字小文字が区別されてソートされます。

WinForms FlexGrid の列のソート順を指定する方法については、次のコードを参照してください。

C#

```
C1.Win.C1FlexGrid.SortFlags order = C1.Win.C1FlexGrid.SortFlags.Ascending |
C1.Win.C1FlexGrid.SortFlags.IgnoreCase;
c1FlexGrid1.Sort(order, 2);
```

VB.NET

```
Dim order As C1.Win.C1FlexGrid.SortFlags = C1.Win.C1FlexGrid.SortFlags.Ascending Or
C1.Win.C1FlexGrid.SortFlags.IgnoreCase
c1FlexGrid1.Sort(order, 2)
```

カスタムソート

FlexGrid には、よく使用されるシナリオに必要なソートオプション（大文字小文字を区別しない、表示値を使用するなど）がいくつか用意されています。ただし、ソート操作の柔軟性を高めたり、ソート操作を詳細に制御する必要がある場合は、IComparer クラスを使用して、カスタムロジックを記述することもできます。たとえば、次の例では、[Name]列をファイル拡張子でソートしています。このサンプルコードでは、ファイル拡張子でソートするためのカスタムロジックが FileNameComparer クラスで実装され、これが **C1FlexGrid** クラスの **Sort()** メソッドにパラメータとして渡されます。

次のコードは、WinForms FlexGrid の列にカスタムソートを適用する方法を示しています。

C#

```
c1FlexGrid1.Sort(new FileNameComparer(c1FlexGrid1, e.Order));

class FileNameComparer : IComparer
 {
   C1FlexGrid c1FlexGrid1;
   bool _desc;

   // ctor
   public FileNameComparer(C1FlexGrid flex, SortFlags order)
   {
    c1FlexGrid1 = flex;
    _desc = ((order & SortFlags.Descending) != 0);
   }

   // IComparer
   public int Compare(object r1, object r2)
   {
     // ファイル名を取得します
     string s1 = (string)c1FlexGrid1[((Row)r1).Index, "Name"];
     string s2 = (string)c1FlexGrid1[((Row)r2).Index, "Name"];

     // 拡張機能を比較します
     int icmp = string.Compare(Path.GetExtension(s1), Path.GetExtension(s2), true);

     // ソート順（昇順または降順）を返します
     return (_desc)? -icmp: icmp;
   }
 }
```

VB.NET

```
C1FlexGrid1.Sort(New FileNameComparer(C1FlexGrid1, SortFlags.Ascending))

Public Class FileNameComparer
    Implements IComparer

    Private c1FlexGrid1 As C1FlexGrid
    Private _desc As Boolean

    Public Sub New(ByVal flex As C1FlexGrid, ByVal order As SortFlags)
        c1FlexGrid1 = flex
        _desc = ((order And SortFlags.Descending) <> 0)
    End Sub

    Public Function Compare(r1 As Object, r2 As Object) As Integer Implements
IComparer.Compare
        Dim s1 As String = CStr(c1FlexGrid1((CType(r1, Row)).Index, "Name"))
        Dim s2 As String = CStr(c1FlexGrid1((CType(r2, Row)).Index, "Name"))
        Dim icmp As Integer = String.Compare(Path.GetExtension(s1),
Path.GetExtension(s2), True)
        Return If((_desc), -icmp, icmp)
    End Function
```

カスタムソートの実装方法の詳細については、**Custom Sort（カスタムソート）**という名前の製品サンプルを参照してください。

メモ: ComponentOneControlPanel.exe を使用して WinForms Edition をインストールする際にサンプルをインストールした場合、前述の製品サンプルは、システムの \Documents\ComponentOne Samples\WinForms\vx.x.x\C1FlexGrid\CS にあります。

ソートインジケータ

FlexGrid には、ソートの方向を示すソートインジケータ（小さな三角矢印の記号）が表示されます。また、グリッドでは、インジケータの表示/非表示の切り替え、位置の指定、カスタマイズなどを柔軟に行うことができます。

ソートインジケータの表示/非表示

デフォルトでは、FlexGrid で列ヘッダーをクリックして列をソートすると、ソートインジケータが表示されます。ただし、ShowSortPosition プロパティを **None** に設定することで、インジケータを非表示にすることもできます。このプロパティは、ShowSortPositionEnum 列挙に含まれる値を受け取ります。

ソートされた WinForms FlexGrid 列に表示されるソートインジケータを非表示にするには、次のコードを使用します。

C#

```
// ソートインジケータを非表示にします
c1FlexGrid1.ShowSortPosition = ShowSortPositionEnum.None;
```

VB.NET

```
' ソートインジケータを非表示にします
c1FlexGrid1.ShowSortPosition = ShowSortPositionEnum.None
```

ソートインジケータの配置

デフォルトでは、ソートインジケータはヘッダーセルの右側に表示されます。ただし、列にフィルタが適用されると、ソートインジケータはヘッダーセルの上部に移動し、空いた場所にフィルタアイコンが表示されます。ソートインジケータの位置を固定するには、**ShowSortPosition** プロパティの値を **Top** または **Right** に設定します。

WinForms FlexGrid の列のソートインジケータの位置をカスタマイズするには、次のコードを使用します。

C#

```
// ソートインジケータの位置を上側に変更します
c1FlexGrid1.ShowSortPosition = ShowSortPositionEnum.Top;
```

VB.NET

```
' ソートインジケータの位置を上側に変更します
c1FlexGrid1.ShowSortPosition = ShowSortPositionEnum.Top
```

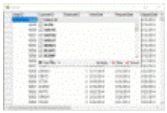
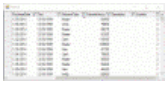
ソートインジケータのカスタマイズ

FlexGrid には、ソートインジケータに使用する画像を指定するための GlyphEnum 列挙が用意されています。**GlyphEnum** とグリフのカスタマイズの詳細については、「カスタムグリフ」を参照してください。

フィルタ

グリッドのフィルタ処理を使用して、列に適用された特定の条件に従ってレコードの表示を絞り込むことができます。この機能は、大規模なデータセットを使用する場合に特に便利です。特定のタイプのレコードのみを表示して、データを簡単に分析できます。

このセクションでは、フィルタの処理、タイプ、およびカスタマイズについて説明します。

トピック	スナップショット	コンテンツ
フィルタ操作		フィルタを有効にする方法、適用する方法、およびその他のフィルタに関連する操作について説明します。 • フィルタの許可 • コードによるフィルタ • カスタムフィルタ処理 • フィルタの削除 • フィルタのタイプ
フィルタの UI		フィルタアイコンとそのカスタマイズ、およびフィルタ言語の設定方法について説明します。 • フィルタアイコンの表示 • カスタムアイコンの使用 • フィルタ言語の変更

フィルタ操作

次の 2 つの方法でグリッドデータのフィルタ処理を実行できます。

ヘッダーベースのフィルタ

ヘッダーベースのフィルタ処理では、列のヘッダーセルにアイコンが表示されます。このアイコンをクリックすると、フィルタを指定するためのオプションがドロップダウンに表示されます。また、特定の列にフィルタが適用されているかどうかも示されます。FlexGrid では、グリッドの AllowFiltering プロパティを使用するだけで、この動作を実行できます。この場合、フィルタのための余分な画面領域は必要ありません。また、この方法では、フィルタエディタを自由にカスタマイズして作成できます。カスタマイズの詳細については、後述のセクションで説明しています。

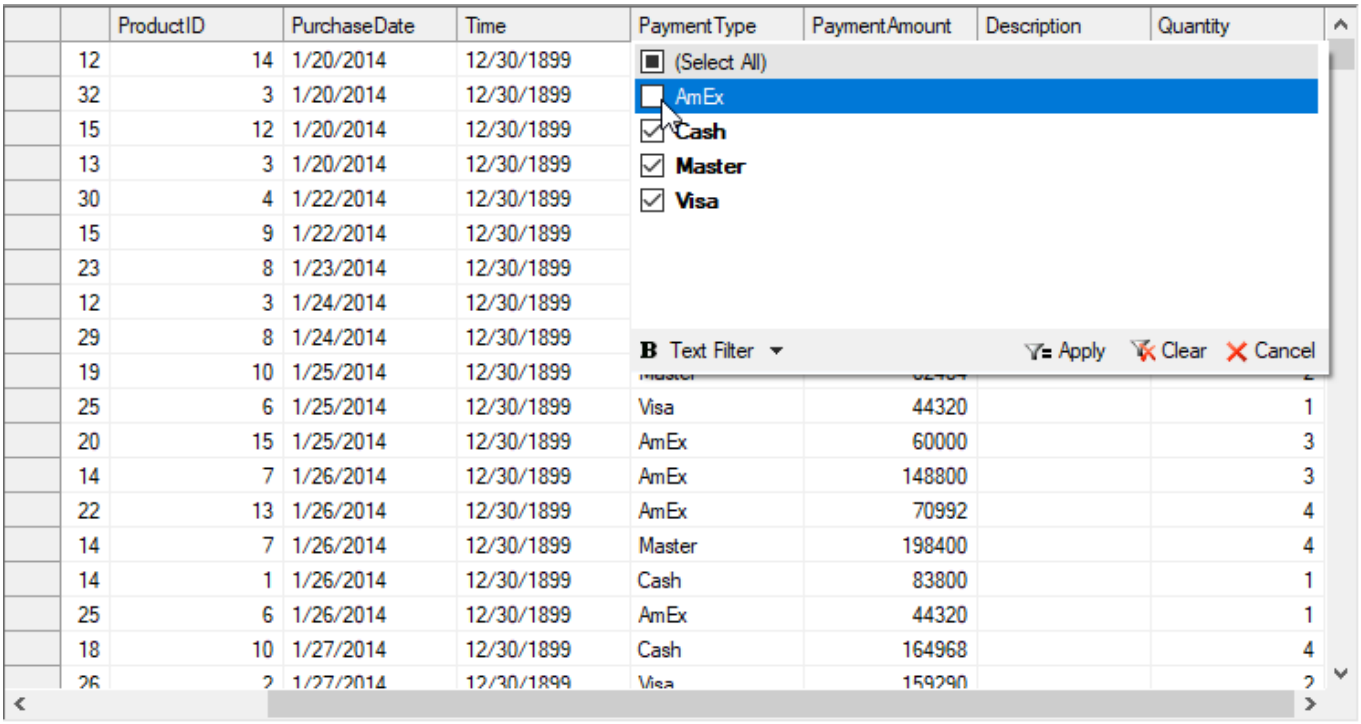
フィルタ行

フィルタ行はフィルタ条件が表示される行で、列ヘッダーの直下に表示されます。この場合、ユーザーは、フィルタ処理された列と現在のフィルタ条件を常に見ることができます。ただし、この長所の半面、フィルタ行のための画面領域が余分に必要になります。FlexGrid では、カスタム実装を使用して、フィルタ行を簡単を追加できます。実装については、「**Filter Row (フィルタ行)**」という名前の製品サンプルを参照してください。

> **メモ**: **ComponentOneControlPanel.exe** を使用して WinForms Edition をインストールする際にサンプルをインストールした場合、「**フィルタ行**」サンプルは、システムの \Documents\ComponentOne Samples\WinForms\v4.5.2\C1FlexGrid\CS にあります。

フィルタの許可

FlexGrid でフィルタ処理を有効にするには、**C1FlexGrid.AllowFiltering** プロパティを **true** に設定します。これで、グリッドのすべての列でデフォルトの ColumnFilter が有効になります。ColumnFilter が有効な場合、ユーザーは実行時に ValueFilter または ConditionFilter を選択できます。



また、各 **Column** オブジェクトの **AllowFiltering** プロパティを設定することで、各列にフィルタタイプを指定することもできます。Column の AllowFiltering プロパティには、次のいずれかの値を選択できます。

値	説明
Default	ValueFilter と ConditionFilter を組み合わせた ColumnFilter が有効になります。ユーザーは、実行時にいずれかを選択できます。
ByValue	その列に含まれる値のチェックボックスリストを含む ValueFilter が有効になります。ユーザーは、それらのチェックボックスをオフにして、フィルタ後の出力に表示しない値を指定できます。
ByCondition	「equals」、「greater than」、「contains」などから 2 つの条件を組み合わせて指定できる ConditionFilter が有効になります。条件を組み合わせるには、「And」または「Or」の演算子を使用します。
Custom	カスタムフィルタをインスタンス化し、それを列の **Filter** プロパティに明示的に割り当てることができます。
None	列のフィルタを無効にします。

WinForms FlexGrid の最初の列で条件フィルタを有効にするには、次のコードスニペットを使用します。

C#

```
// グリッドでのフィルタリングを許可します
c1FlexGrid1.AllowFiltering = true;

// 列1に対してフィルタを指定します
c1FlexGrid1.Cols[1].AllowFiltering = C1.Win.C1FlexGrid.AllowFiltering.ByCondition;
```

VB.NET

```
' グリッドでのフィルタリングを許可します
c1FlexGrid1.AllowFiltering = True
' 列1に対してフィルタを指定します
c1FlexGrid1.Cols(1).AllowFiltering = C1.Win.C1FlexGrid.AllowFiltering.ByCondition
```

前述のフィルタタイプの詳細については、「フィルタのタイプ」を参照してください。

コードによるフィルタ

グリッドの AllowFiltering プロパティを指定すると、グリッドのフィルタ処理が有効になります。このプロパティは多くの一般的なシナリオで使用できますが、フィルタオプションの調整がさらに必要な場合もあります。それには、各列の **AllowFiltering** プロパティと **Filter** プロパティを変更します。たとえば、次のコードは、WinForms FlexGrid のフィルタ処理を文字列型の列に制限して有効にしています。

C#

```
// 文字列型の列にフィルタリングを制限します
 foreach (Column c in c1FlexGrid1.Cols)
     {
          c.AllowFiltering = c.DataType == typeof(string)
               ? AllowFiltering.Default
               : AllowFiltering.None;
     }
```

VB.NET

```
' 文字列型の列にフィルタリングを制限します
For Each c As Column In c1FlexGrid1.Cols
    c.AllowFiltering = If(c.DataType Is GetType(String), AllowFiltering.Default,
AllowFiltering.None)
Next
```

さらに、フィルタを作成して列に割り当てたり、既存のフィルタを取得してそのプロパティを変更することで、フィルタ処理をカスタマイズすることもできます。たとえば、次のコードでは、WinForms FlexGrid に **ConditionFilter** を作成して、「Germany」に一致するすべての項目を選択するように設定し、この新しいフィルタを[ShipCountry]列に割り当てています。

C#

```
// グリッドでフィルタリングを有効にします
 c1FlexGrid1.AllowFiltering = true;

 // 新しいConditionFilterを作成します
 C1.Win.C1FlexGrid.ConditionFilter Filter;
 Filter = new C1.Win.C1FlexGrid.ConditionFilter();

 // Germanyのアイテムを絞り込むためのフィルタを作成します
 Filter.Condition1.Operator = C1.Win.C1FlexGrid.ConditionOperator.Equals;
 Filter.Condition1.Parameter = "Germany";

 //「ShipCountry」列に新しいフィルタを割り当てます
 c1FlexGrid1.Cols["ShipCountry"].Filter = Filter;

 // フィルタを適用します
 c1FlexGrid1.ApplyFilters();
```

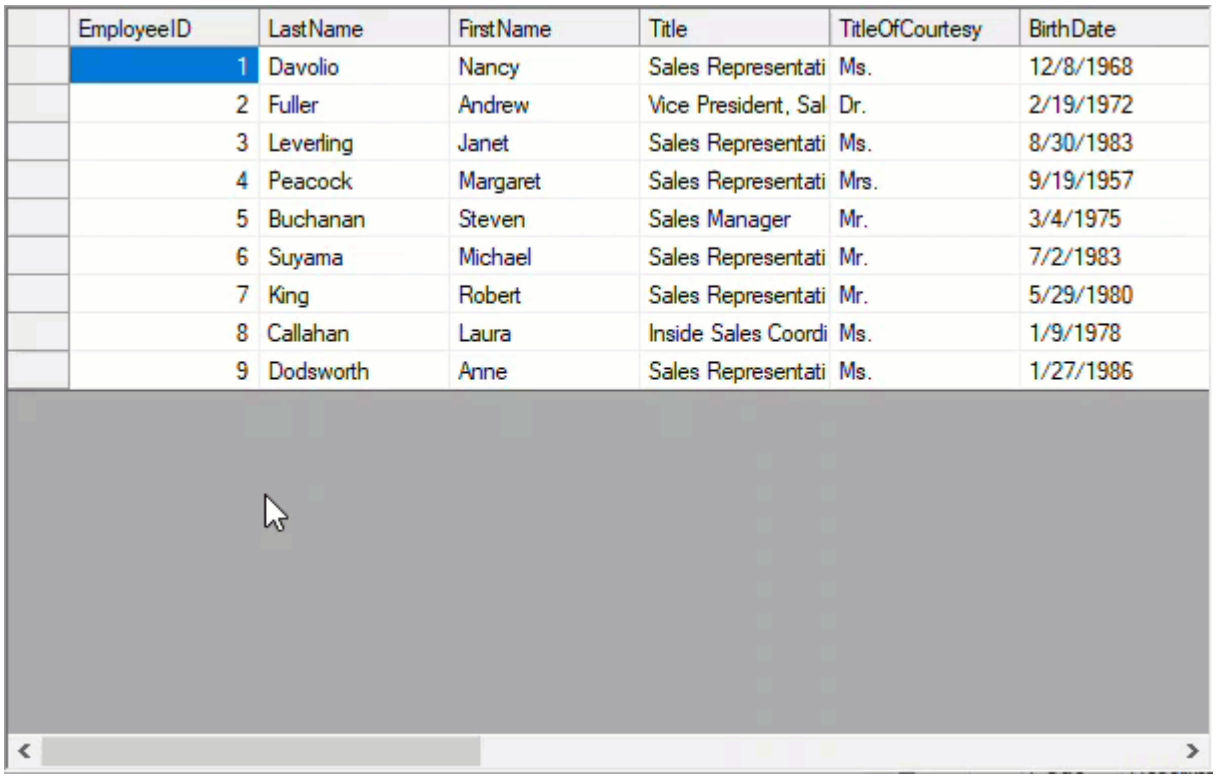
VB.NET

```
' グリッドでフィルタリングを有効にします
c1FlexGrid1.AllowFiltering = True
' 新しいConditionFilterを作成します
Dim Filter As C1.Win.C1FlexGrid.ConditionFilter
Filter = New C1.Win.C1FlexGrid.ConditionFilter()
' Germanyのアイテムを絞り込むためのフィルタを作成します
Filter.Condition1.[Operator] = C1.Win.C1FlexGrid.ConditionOperator.Equals
Filter.Condition1.Parameter = "Germany"
'「ShipCountry」列に新しいフィルタを割り当てます
```

```
c1FlexGrid1.Cols("ShipCountry").Filter = Filter
' フィルタを適用します
c1FlexGrid1.ApplyFilters()
```

カスタムフィルタ処理

FlexGrid にフィルタ処理を適用すると、フィルタ条件を満たさない行は、Visible プロパティが **false** に設定されて非表示になります。ただし、この動作のカスタマイズが必要な場合もあります。そのようなシナリオでは、FlexGrid が発生する **BeforeFilter** イベントと **AfterFilter** イベントを利用できます。次のサンプルでは、条件を満たさない行を非表示にする代わりに、それらの行に別のスタイルを適用して、WinForms FlexGrid のフィルタ処理動作をカスタマイズしています。



C#

```
private void c1FlexGrid1_BeforeFilter(object sender, CancelEventArgs e)
{
    c1FlexGrid1.BeginUpdate();
}

private void c1FlexGrid1_AfterFilter(object sender, EventArgs e)
{
    // フィルタリングされた行を表示するために使用されるスタイルを取得します
    var cs = c1FlexGrid1.Styles["filteredOut"];

    // すべての行にスタイルを適用します
    for (int r = c1FlexGrid1.Rows.Fixed; r < c1FlexGrid1.Rows.Count; r++)
    {
        var row = c1FlexGrid1.Rows[r];
        if (row.Visible)
        {
            // 通常の行、スタイルをリセットします
            row.Style = null;
        }
        else
        {
            // フィルタリングされた行。 見えるようにし、スタイルを適用します。
```

```
            row.Visible = true;
            row.Style = cs;
        }
    }

    // 更新を再開します
    c1FlexGrid1.EndUpdate();
}
```

VB.NET

```
Private Sub c1FlexGrid1_BeforeFilter(ByVal sender As Object, ByVal e As
CancelEventArgs)
    c1FlexGrid1.BeginUpdate()
End Sub

Private Sub c1FlexGrid1_AfterFilter(ByVal sender As Object, ByVal e As EventArgs)
    ' フィルタリングされた行を表示するために使用されるスタイルを取得します
    Dim cs = c1FlexGrid1.Styles("filteredOut")

    ' すべての行にスタイルを適用します
    For r = c1FlexGrid1.Rows.Fixed To c1FlexGrid1.Rows.Count - 1
        Dim row = c1FlexGrid1.Rows(r)

        If row.Visible Then
            ' 通常の行、スタイルをリセットします
            row.Style = Nothing
        Else
            ' フィルタリングされた行。 見えるようにし、スタイルを適用します。
            row.Visible = True
            row.Style = cs
        End If
    Next

    ' 更新を再開します
    c1FlexGrid1.EndUpdate()
End Sub
```

前述のコードでは、「filteredout」というカスタムスタイルを使用しています。これは、次のコードで定義されます。

C#

```
// フィルタによって除外された行のスタイルを作成します
var cs = c1FlexGrid1.Styles.Add("filteredOut");
cs.BackColor = Color.LightGray;
cs.ForeColor = Color.DarkGray;
```

VB.NET

```
' フィルタによって除外された行のスタイルを作成します
Dim cs = c1FlexGrid1.Styles.Add("filteredOut")
cs.BackColor = Color.LightGray
cs.ForeColor = Color.DarkGray
```

フィルタの削除

ユーザーは、各列のフィルタ UI にある[**クリア**]オプションを使用して、実行時に列フィルタ処理を削除できます。また、グリッド全体のフィルタ処理をプログラムで削除することもできます。それには、FlexGrid の **FilterDefinition** プロパティに空の文字列を渡します。

次のコードは、WinForms FlexGrid からすべてのフィルタをクリアする方法を示しています。

C#

```
// フィルタ定義を空の文字列に設定して、フィルタリングを削除します
c1FlexGrid1.FilterDefinition = string.Empty;
```

VB.NET

```
' フィルタ定義を空の文字列に設定して、フィルタリングを削除します
c1FlexGrid1.FilterDefinition = String.Empty
```

フィルタのタイプ

FlexGrid では、列フィルタ、値フィルタ、条件フィルタなどの組み込みフィルタを使用できるだけでなく、独自のカスタムフィルタを作成することもできます。組み込みフィルタは Column クラスの **AllowFiltering** プロパティを指定することで提供されます。このプロパティで、特定の列に適用するフィルタのタイプを設定します。一方、カスタムフィルタでは IC1ColumnFilter インタフェースと IC1ColumnFilterEditor インタフェースを使用します。このトピックでは、組み込みフィルタとカスタムフィルタの実装について詳しく説明します。

列フィルタ

列フィルタは、グリッドの **AllowFiltering** プロパティが **true** に設定されている場合に、すべての列に自動的に適用されるデフォルトのフィルタです。ColumnFilter は ValueFilter と ConditionFilter を組み合わされたもの（後述のセクションを参照）、ユーザーは実行時にこのどちらかを選択することができます。コードを使用する場合は、**ValueFilter** プロパティと **ConditionFilter** プロパティを使用して、この 2 つのタイプのフィルタにアクセスします。また、フィルタを値に適用する **Apply** メソッドと、フィルタをリセットして非アクティブ化する **Reset** メソッドが用意されています。

	ProductID	PurchaseDate	Time	PaymentType	PaymentAmount	Description	Quantity
12	14	1/20/2014	12/30/1899	Master	64000		5
32	3	1/20/2014	12/30/1899	AmEx	76800		3
15	12	1/20/2014	12/30/1899	Master	86575		5
13	3	1/20/2014	12/30/1899	Master	51200		2
30	4	1/22/2014	12/30/1899	Cash	118350		3
15	9	1/22/2014	12/30/1899	Master	109800		2
23	8	1/23/2014	12/30/1899	Visa	47780		1
12	3	1/24/2014	12/30/1899	Cash	76800		3
29	8	1/24/2014	12/30/1899	Master	95560		2
19	10	1/25/2014	12/30/1899	Master	82484		2
25	6	1/25/2014	12/30/1899	Visa	44320		1
20	15	1/25/2014	12/30/1899	AmEx	60000		3
14	7	1/26/2014	12/30/1899	AmEx	148800		3
22	13	1/26/2014	12/30/1899	AmEx	70992		4
14	7	1/26/2014	12/30/1899	Master	198400		4
14	1	1/26/2014	12/30/1899	Cash	83800		1
25	6	1/26/2014	12/30/1899	AmEx	44320		1
18	10	1/27/2014	12/30/1899	Cash	164968		4
26	2	1/27/2014	12/30/1899	Visa	159290		2

WinForms FlexGrid の[ProductName]列に列フィルタを適用するには、次のコードを使用します。

C#

```
// 列フィルタ(ProductNameが「C」で始まる製品をフィルター処理します)
ColumnFilter colFilter = new ColumnFilter();
colFilter.ConditionFilter.Condition1.Parameter = "C";
```

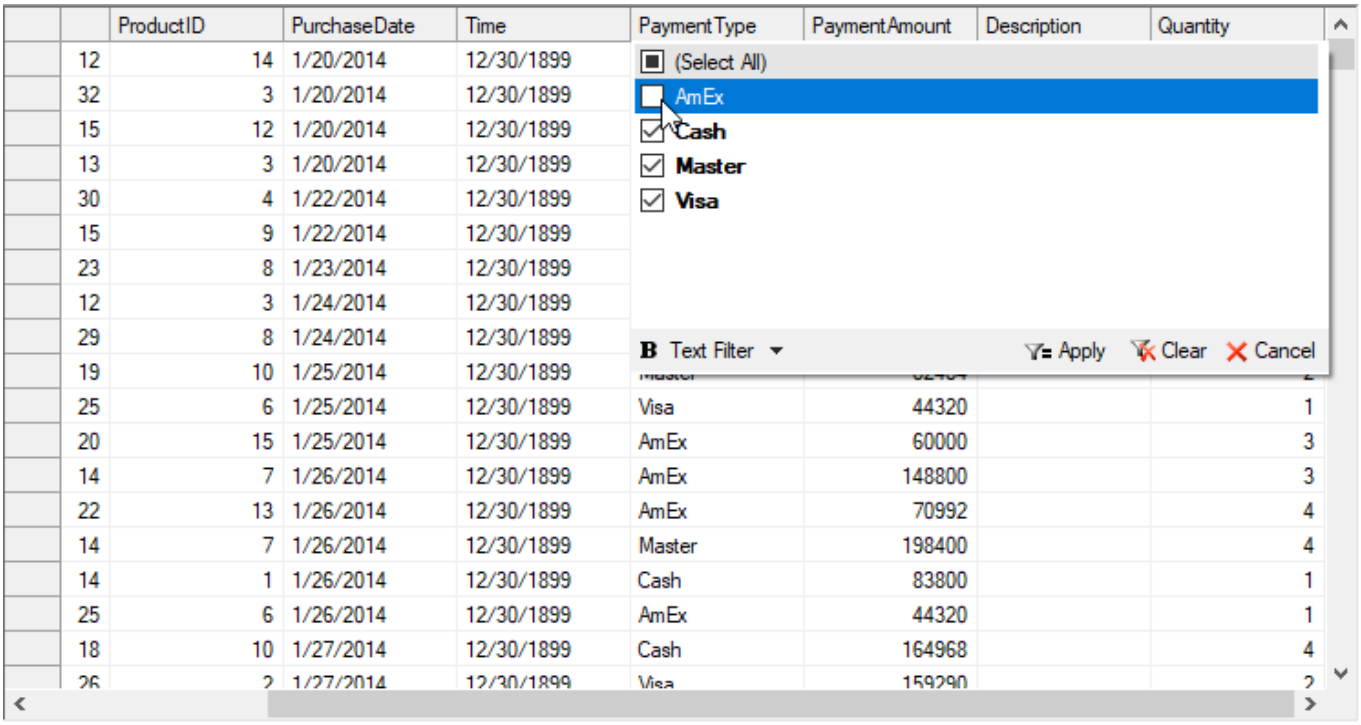
```
colFilter.ConditionFilter.Condition1.Operator = ConditionOperator.BeginsWith;
c1FlexGrid1.Cols["ProductName"].Filter = colFilter;
```

VB.NET

```
' 列フィルタ(ProductNameが「C」で始まる製品をフィルター処理します)
Dim colFilter As ColumnFilter = New ColumnFilter()
colFilter.ConditionFilter.Condition1.Parameter = "C"
colFilter.ConditionFilter.Condition1.[Operator] = ConditionOperator.BeginsWith
c1FlexGrid1.Cols("ProductName").Filter = colFilter
```

値フィルタ

値フィルタは値ベースのフィルタで、列の AllowFiltering プロパティを **ByValue** に設定することで有効になります。ValueFilter では、すべての値がチェックボックス付きでドロップダウンリストに表示されます。ユーザーは、チェックボックスを使用して、出力に表示する値を選択できます。これらの値を取得または設定するには **ShowValues** プロパティを使用します。また、**ValuesLimit** プロパティを設定して、ドロップダウンリストに表示する値の数を制限することもできます。列フィルタと同様に、値フィルタにも値にフィルタを適用する **Apply** メソッドと、フィルタをリセットする **Reset** メソッドが用意されています。



次のコードは、WinForms FlexGrid の列に ValueFilter を適用する方法を示しています。

C#

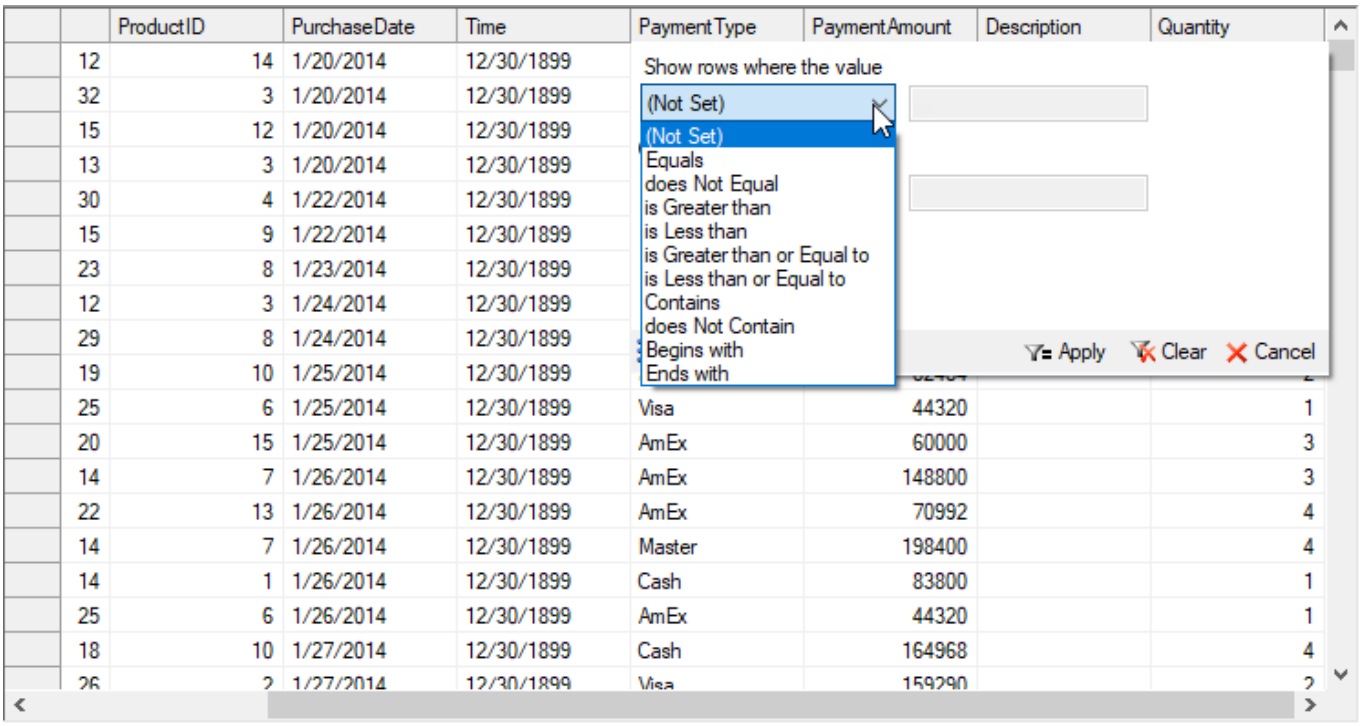
```
// 値フィルタ(CategoryIdが1,2,3,5,8である製品をフィルタ処理します)
ValueFilter valueFilter = new ValueFilter();
valueFilter.ShowValues = new object[] { 1, 2, 3, 5, 8 };
c1FlexGrid1.Cols["CategoryId"].Filter = valueFilter;
```

VB.NET

```
' 値フィルタ(CategoryIdが1,2,3,5,8である製品をフィルタ処理します)
Dim valueFilter As ValueFilter = New ValueFilter()
valueFilter.ShowValues = New Object() {1, 2, 3, 5, 8}
c1FlexGrid1.Cols("CategoryId").Filter = valueFilter
```

条件フィルタ

条件フィルタは、1 つまたは 2 つの論理条件に基づいたフィルタで、**AllowFiltering** プロパティを **ByCondition** に設定することで有効にできます。**ConditionFilter** では、**Condition1** プロパティと **Condition2** プロパティを通して、ユーザーに 1 つまたは 2 つの条件を設定するオプションを表示できます。これらのプロパティを AND または OR 演算子で結合してレコードをフィルタするには、**AndConditions** プロパティを設定します。他のフィルタと同様に、このクラスにも **Apply** メソッドと **Reset** メソッドが用意されています。



次のコードは、WinForms FlexGrid の列に条件フィルタを適用する方法を示しています。

C#

```
// 条件フィルタ(UnitPrice> = 50および<= 100の製品をフィルタ処理します)
ConditionFilter conditionFilter = new ConditionFilter();
conditionFilter.Condition1.Parameter = 50;
conditionFilter.Condition1.Operator = ConditionOperator.GreaterThanOrEqualTo;
conditionFilter.Condition2.Parameter = 100;
conditionFilter.Condition2.Operator = ConditionOperator.LessThanOrEqualTo;
conditionFilter.AndConditions = true;
c1FlexGrid1.Cols["UnitPrice"].Filter = conditionFilter;
```

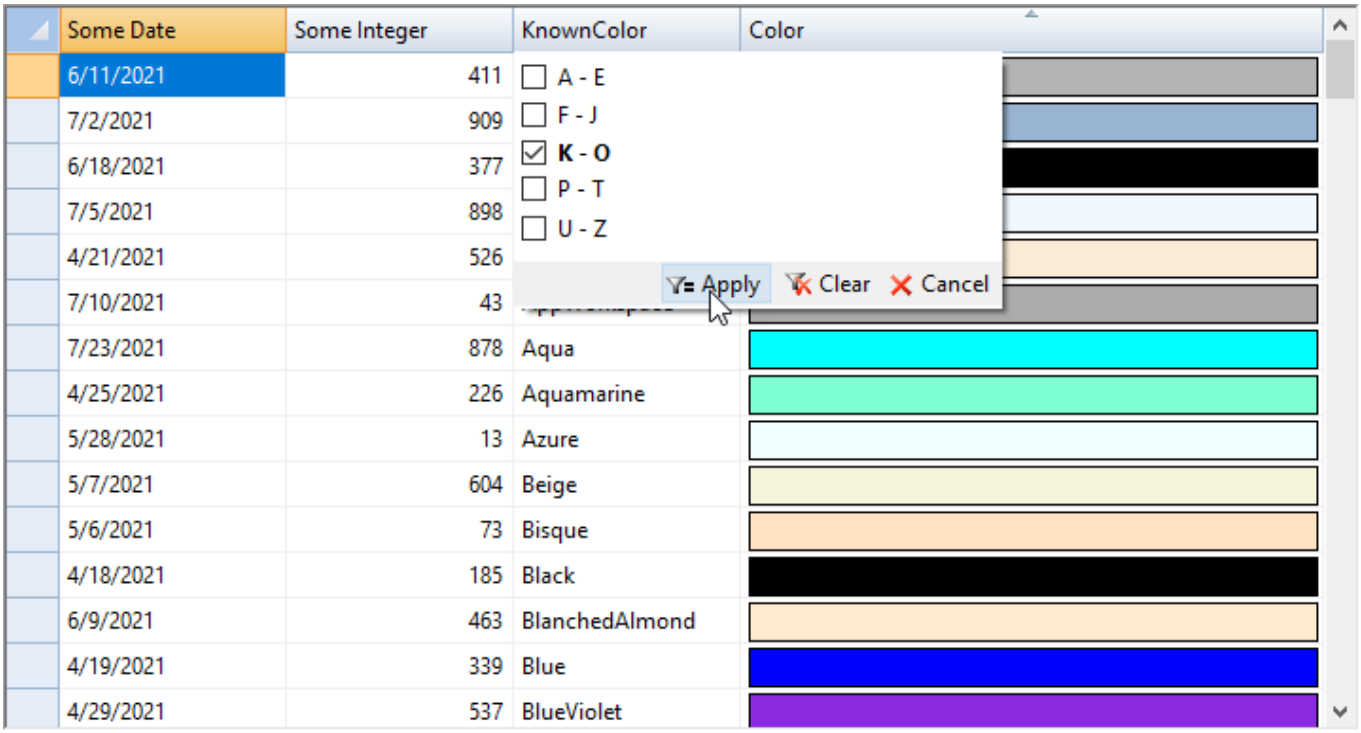
VB.NET

```
' 条件フィルタ(UnitPrice> = 50および<= 100の製品をフィルタ処理します)
Dim conditionFilter As ConditionFilter = New ConditionFilter()
conditionFilter.Condition1.Parameter = 50
conditionFilter.Condition1.[Operator] = ConditionOperator.GreaterThanOrEqualTo
conditionFilter.Condition2.Parameter = 100
conditionFilter.Condition2.[Operator] = ConditionOperator.LessThanOrEqualTo
conditionFilter.AndConditions = True
c1FlexGrid1.Cols("UnitPrice").Filter = conditionFilter
```

カスタムフィルタ

前述のフィルタは、多くの一般的なフィルタ処理シナリオを効率的に実装するのに十分な柔軟性を備えています。ただし、カス

タムフィルタオプションを使用して、ほかにもアプリケーションの特殊な要件を満たす独自のフィルタを作成することもできます。カスタムフィルタを作成するには、IC1ColumnFilter インタフェースを実装するフィルタクラス、および IC1ColumnFilterEditor インタフェースを実装するエディタクラスを作成する必要があります。次の例は、範囲に基づいてフィルタできるカスタム文字列フィルタの実装を示しています。



次のコードは、WinForms FlexGrid 向けに作成されたカスタムフィルタのコードです。このクラスでは **IC1CustomFilter** インタフェースを実装しています。

StringFilter.cs

C#

```
class StringFilter : C1.Win.C1FlexGrid.IC1ColumnFilter
{

    List<Point> _ranges = new List<Point>();

    /// <summary>
    /// このフィッタが受け入れる文字列を定義するポイントのリストを取得します
    /// </summary>
    public List<Point> Ranges
    {
        get { return _ranges; }
    }


    // 範囲リストが空でない場合、フィルタはアクティブです
    public bool IsActive
    {
        get { return _ranges.Count > 0; }
    }

    // フィルタをリセットします
    public void Reset()
    {
        _ranges.Clear();
    }
```

```
    // 指定された日付にフィルタを適用します
    public bool Apply(object value)
    {
        if (value != null)
        {
            var s = value.ToString();
            if (s.Length > 0)
            {
                int c = char.ToUpperInvariant(s[0]);
                foreach (var cr in _ranges)
                {
                    if (c >= char.ToUpperInvariant((char)cr.X) &&
                        c <= char.ToUpperInvariant((char)cr.Y))
                    {
                        return true;
                    }
                }
            }
        }
        return false;
    }

    // このフィルタのエディタコントロールを返します
    public C1.Win.C1FlexGrid.IC1ColumnFilterEditor GetEditor()
    {
        return new StringFilterEditor();
    }

}
```

VB.NET

```
Friend Class StringFilter
    Implements C1.Win.C1FlexGrid.IC1ColumnFilter
    Private _ranges As List(Of Point) = New List(Of Point)()


    ''' <summary>
    ''' このフィッタが受け入れる文字列を定義するポイントのリストを取得します
    ''' </summary>
    Public ReadOnly Property Ranges As List(Of Point)
        Get
            Return _ranges
        End Get
    End Property

    ' 範囲リストが空でない場合、フィルタはアクティブです
    Public ReadOnly Property IsActive As Boolean Implements
C1.Win.C1FlexGrid.IC1ColumnFilter.IsActive
        Get
            Return _ranges.Count > 0
        End Get
    End Property

    ' フィルタをリセットします
    Public Sub Reset() Implements C1.Win.C1FlexGrid.IC1ColumnFilter.Reset
        _ranges.Clear()
    End Sub

    ' 指定された日付にフィルタを適用します
    Public Function Apply(ByVal value As Object) As Boolean Implements
C1.Win.C1FlexGrid.IC1ColumnFilter.Apply
```

```
        If value IsNot Nothing Then
            Dim s = value.ToString()

            If s.Length > 0 Then
                Dim c As Integer = AscW(Char.ToUpperInvariant(s(0)))

                For Each cr In _ranges

                    If c >=
AscW(Char.ToUpperInvariant(Microsoft.VisualBasic.ChrW(cr.X))) AndAlso c <=
AscW(Char.ToUpperInvariant(Microsoft.VisualBasic.ChrW(cr.Y))) Then
                        Return True
                    End If
                Next
            End If
        End If

        Return False
    End Function

    ' このフィルタのエディタコントロールを返します
    Public Function GetEditor() As C1.Win.C1FlexGrid.IC1ColumnFilterEditor
Implements C1.Win.C1FlexGrid.IC1ColumnFilter.GetEditor
        Return New StringFilterEditor()
    End Function

End Class
```

次のコードは、**IC1CustomFilterEditor** インタフェースを実装するカスタムクラスのコードです。

StringFilterEditor.cs

C#

```
public partial class StringFilterEditor :
    UserControl,
    C1.Win.C1FlexGrid.IC1ColumnFilterEditor
{
    StringFilter _filter;

    public StringFilterEditor()
    {
        InitializeComponent();
    }

    public void Initialize(C1.Win.C1FlexGrid.C1FlexGridBase grid, int columnIndex,
C1.Win.C1FlexGrid.IC1ColumnFilter filter)
    {
        _filter = (StringFilter)filter;

        // チェックボックスの値を初期化します
        foreach (var pt in _filter.Ranges)
        {
            switch ((char)pt.X)
            {
                case 'A':
                    _chkAE.Checked = true;
                    break;
                case 'F':
                    _chkFJ.Checked = true;
                    break;
```

```
                case 'K':
                    _chkKO.Checked = true;
                    break;
                case 'P':
                    _chkPT.Checked = true;
                    break;
                case 'U':
                    _chkUZ.Checked = true;
                    break;
            }
        }
    }
    public void ApplyChanges()
    {
        // フィルタをリセットします
        _filter.Ranges.Clear();

        // 選択した範囲を追加します
        foreach (Control ctl in this.Controls)
        {
            var cb = ctl as CheckBox;
            if (cb != null && cb.Checked)
            {
                var pt = new Point((int)cb.Text[0], (int)cb.Text[cb.Text.Length -
1]);
                _filter.Ranges.Add(pt);
            }
        }
    }
    public bool KeepFormOpen
    {
        get { return false; }
    }

    void _chkAE_CheckedChanged(object sender, EventArgs e)
    {
        var cb = sender as CheckBox;
        cb.Font = new Font(Font, cb.Checked ? FontStyle.Bold : FontStyle.Regular);
    }
}
```

VB.NET

```
Public Partial Class StringFilterEditor
    Inherits UserControl
    Implements C1.Win.C1FlexGrid.IC1ColumnFilterEditor

    Private _filter As StringFilter
    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub Initialize(ByVal grid As C1.Win.C1FlexGrid.C1FlexGridBase, ByVal
columnIndex As Integer, ByVal filter As C1.Win.C1FlexGrid.IC1ColumnFilter)
Implements C1.Win.C1FlexGrid.IC1ColumnFilterEditor.Initialize
        _filter = CType(filter, StringFilter)

        ' チェックボックスの値を初期化します
        For Each pt In _filter.Ranges

            Select Case Microsoft.VisualBasic.ChrW(pt.X)
                Case "A"c
                    _chkAE.Checked = True
```

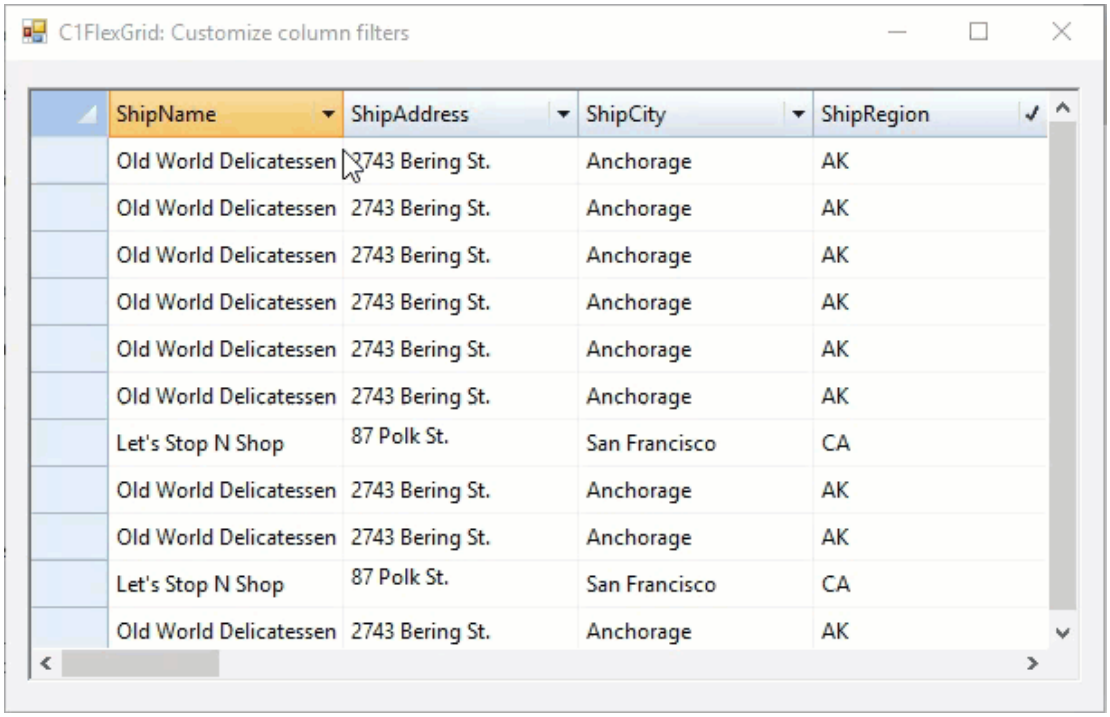
```
                Case "F"c
                    _chkFJ.Checked = True
                Case "K"c
                    _chkKO.Checked = True
                Case "P"c
                    _chkPT.Checked = True
                Case "U"c
                    _chkUZ.Checked = True
            End Select
        Next
    End Sub

    Public Sub ApplyChanges() Implements
C1.Win.C1FlexGrid.IC1ColumnFilterEditor.ApplyChanges
        ' フィルタをリセットします
        _filter.Ranges.Clear()

        ' 選択した範囲を追加します
        For Each ctl As Control In Controls
            Dim cb = TryCast(ctl, CheckBox)

            If cb IsNot Nothing AndAlso cb.Checked Then
                Dim pt = New Point(Microsoft.VisualBasic.AscW(cb.Text(0)),
Microsoft.VisualBasic.AscW(cb.Text(cb.Text.Length - 1)))
                _filter.Ranges.Add(pt)
            End If
        Next
    End Sub

    Public ReadOnly Property KeepFormOpen As Boolean Implements
C1.Win.C1FlexGrid.IC1ColumnFilterEditor.KeepFormOpen
        Get
            Return False
        End Get
    End Property

    Private Sub _chkAE_CheckedChanged(ByVal sender As Object, ByVal e As EventArgs)
        Dim cb = TryCast(sender, CheckBox)
        cb.Font = New Font(MyBase.Font, If(cb.Checked, FontStyle.Bold,
FontStyle.Regular))
    End Sub

End Class
```

フィルタの UI

FlexGrid では、フィルタ処理 UI を柔軟にカスタマイズできるオプションがいくつか提供されており、アプリケーションに独自のフィルタを作成できます。フィルタアイコンの表示/非表示やカスタマイズを行うことができます。ローカライズ要件に応じて、フィルタに表示される言語を変更することもできます。

フィルタアイコンの表示

デフォルトでは、FlexGrid のフィルタ処理可能な列のヘッダー上にマウスが置かれると、フィルタアイコン( )が表示されます。ただし、C1FlexGrid クラスの **ShowFilterIcon** プロパティを **Always** に設定することで、フィルタアイコンを常に表示することができます。このプロパティは、FilterIconVisibility 列挙に含まれる値を受け取ります。



WinForms FlexGrid のフィルタ処理される列にフィルタアイコンを常に表示するには、次のコードを使用します。

C#

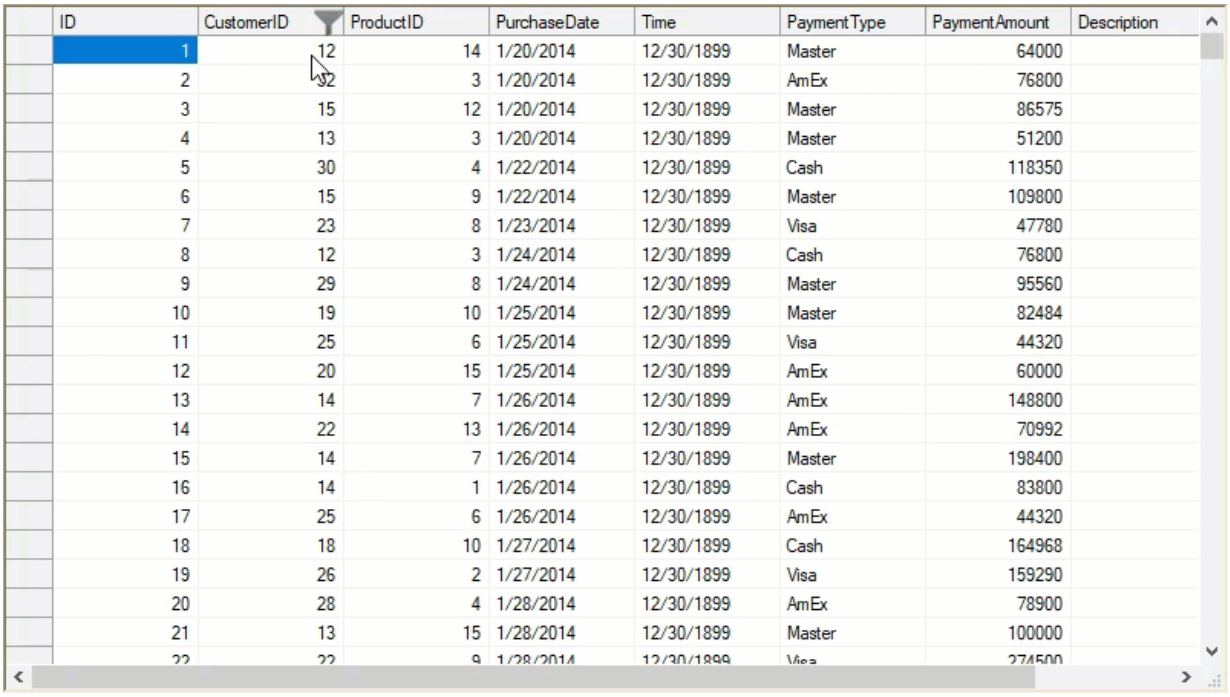
```
// 常にフィルタのアイコンを表示します
c1FlexGrid1.ShowFilterIcon = C1.Win.C1FlexGrid.FilterIconVisibility.Always;
```

VB.NET

```
' 常にフィルタのアイコンを表示します
c1FlexGrid1.ShowFilterIcon = C1.Win.C1FlexGrid.FilterIconVisibility.Always
```

カスタムアイコンの使用

FlexGrid では、GlyphEnum 列挙を使用して画像を受け取る **C1FlexGrid** クラスの Glyphs プロパティを使用して、フィルタアイコンをカスタマイズすることもできます。同じ列挙を使用して、現在フィルタ処理されている列に表示されるアイコンを変更することもできます。カスタムグリフの詳細については、カスタムグリフ を参照してください。



ID	CustomerID	ProductID	PurchaseDate	Time	PaymentType	PaymentAmount	Description
1	12	14	1/20/2014	12/30/1899	Master	64000	
2	32	3	1/20/2014	12/30/1899	AmEx	76800	
3	15	12	1/20/2014	12/30/1899	Master	86575	
4	13	3	1/20/2014	12/30/1899	Master	51200	
5	30	4	1/22/2014	12/30/1899	Cash	118350	
6	15	9	1/22/2014	12/30/1899	Master	109800	
7	23	8	1/23/2014	12/30/1899	Visa	47780	
8	12	3	1/24/2014	12/30/1899	Cash	76800	
9	29	8	1/24/2014	12/30/1899	Master	95560	
10	19	10	1/25/2014	12/30/1899	Master	82484	
11	25	6	1/25/2014	12/30/1899	Visa	44320	
12	20	15	1/25/2014	12/30/1899	AmEx	60000	
13	14	7	1/26/2014	12/30/1899	AmEx	148800	
14	22	13	1/26/2014	12/30/1899	AmEx	70992	
15	14	7	1/26/2014	12/30/1899	Master	198400	
16	14	1	1/26/2014	12/30/1899	Cash	83800	
17	25	6	1/26/2014	12/30/1899	AmEx	44320	
18	18	10	1/27/2014	12/30/1899	Cash	164968	
19	26	2	1/27/2014	12/30/1899	Visa	159290	
20	28	4	1/28/2014	12/30/1899	AmEx	78900	
21	13	15	1/28/2014	12/30/1899	Master	100000	
22	22	9	1/28/2014	12/30/1899	Visa	274500	

次のコードは、WinForms FlexGrid のフィルタ処理される列にカスタムアイコンを適用する方法を示します。

C#

```
// フィルタのアイコンのグリフをカスタマイズします
c1FlexGrid1.Glyphs[GlyphEnum.FilterEditor] = Image.FromFile("custom-filter-icon.png");
// フィルタアイコンのグリフをカスタマイズします
c1FlexGrid1.Glyphs[GlyphEnum.FilteredColumn] = Image.FromFile("filter.ico");
```

VB.NET

```
' フィルタのアイコンのグリフをカスタマイズします
c1FlexGrid1.Glyphs(GlyphEnum.FilterEditor) = Image.FromFile("custom-filter-icon.png")
' フィルタアイコンのグリフをカスタマイズします
c1FlexGrid1.Glyphs(GlyphEnum.FilteredColumn) = Image.FromFile("filter.ico")
```

フィルタ言語の変更

デフォルトでは、FlexGrid の列フィルタエディタは、**CurrentUICulture** 設定で指定されている言語を使用するようにローカライズされます。ただし、**Language** プロパティを使用すると、デフォルトをオーバーライドして、グリッドに列フィルタエディタを表示するときに使用する言語を指定できます。

次のコードを使用して、WinForms FlexGrid でのフィルタの表示言語を変更できます。

C#

```
// フィルタ言語を日本語に設定します
c1FlexGrid1.Language = C1.Util.Localization.Language.Japanese;
```

VB.NET

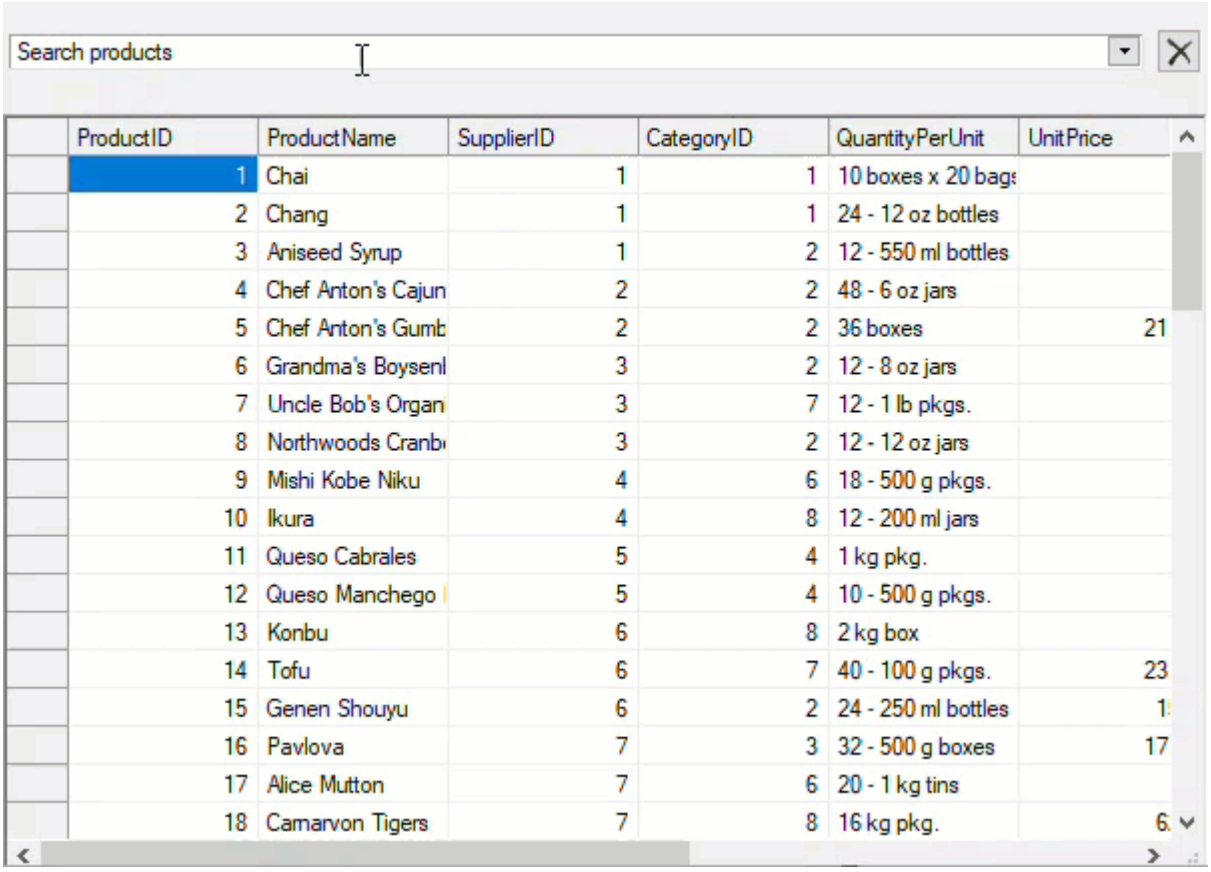
```
' フィルタ言語を日本語に設定します
c1FlexGrid1.Language = C1.Util.Localization.Language.Japanese
```

検索

FlexGrid を使用すると、グリッド全体、または特定の列のみを対象とした検索を簡単に実行できます。このトピックでは、FlexGrid で検索を行う方法について説明します。

グリッド全体の検索

FlexGrid 全体を検索するには、C1FlexGridSearchPanel コントロールをフォームに追加し、**SetC1FlexGridSearchPanel** メソッドを使用して検索対象のグリッドと関連付ける必要があります。C1FlexGridSearchPanel コントロールは、検索結果を強調表示し、検索結果が含まれるレコードを自動的にフィルタ処理します。ただし、**HighlightSearchResults** を **false** に設定することで、結果を強調表示しないようにすることもできます。また、検索を自動的に開始するか、[検索]ボタンまたは[Enter]キーが押されたときに開始するかを設定する **SearchMode** プロパティも提供されています。検索ボックス内の**ウォーターマーク**や、**検索の遅延時間**を設定することもできます。



SearchPanel を使用して WinForms FlexGrid 全体を検索するには、次のコードを使用します。

C#

```
// 検索パネルを検索対象のFlexGridに関連付けます。
c1FlexGridSearchPanel1.SetC1FlexGridSearchPanel(c1FlexGrid1,
c1FlexGridSearchPanel1);
// 検索を構成します
c1FlexGridSearchPanel1.HighlightSearchResults = true;
c1FlexGridSearchPanel1.SearchMode = C1.Win.C1FlexGrid.SearchMode.Always;
c1FlexGridSearchPanel1.Watermark = "Search products";
c1FlexGridSearchPanel1.SearchDelay = 2;
```

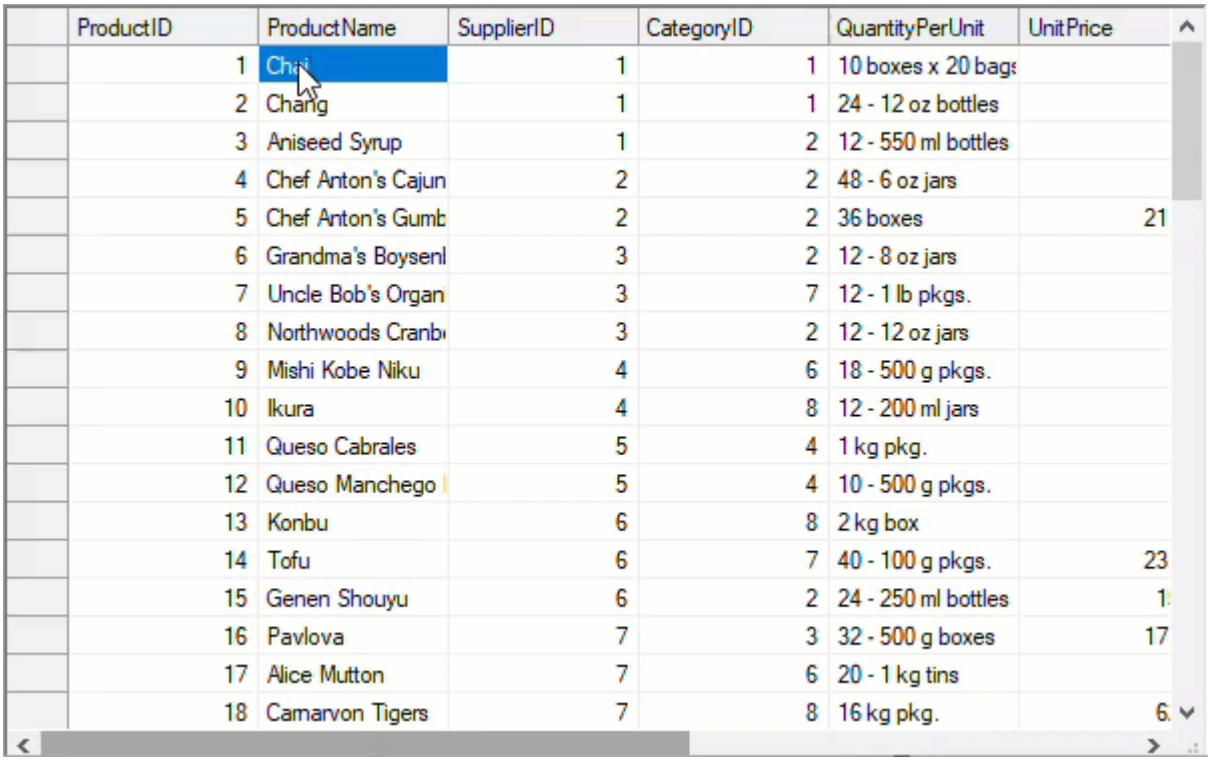
VB.NET

```
' 検索パネルを検索対象のFlexGridに関連付けます。
c1FlexGridSearchPanel1.SetC1FlexGridSearchPanel(c1FlexGrid1, c1FlexGridSearchPanel1)
```

```
' 検索を構成します
c1FlexGridSearchPanel1.HighlightSearchResults = True
c1FlexGridSearchPanel1.SearchMode = C1.Win.C1FlexGrid.SearchMode.Always
c1FlexGridSearchPanel1.Watermark = "Search products"
c1FlexGridSearchPanel1.SearchDelay = 2
```

列内の検索

列方向の検索を可能にするために、FlexGrid には、AutoSearchEnum 列挙の値を受け取る C1FlexGrid クラスの **AutoSearch** プロパティが用意されています。この列挙を使用して、スクロール可能な最初の行、または現在のカーソル位置から下方向への列検索を開始できます。特定の列で値を検索するには、カーソルをその列内に置き、検索する値を入力します。これで、グリッドは、その列内の検索結果に自動的にジャンプします。**AutoSearchDelay** プロパティを設定して、検索の遅延を設定することもできます。



	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice
	1	Chai	1	1	10 boxes x 20 bags	
	2	Chang	1	1	24 - 12 oz bottles	
	3	Aniseed Syrup	1	2	12 - 550 ml bottles	
	4	Chef Anton's Cajun	2	2	48 - 6 oz jars	
	5	Chef Anton's Gumb	2	2	36 boxes	21
	6	Grandma's Boysenl	3	2	12 - 8 oz jars	
	7	Uncle Bob's Organ	3	7	12 - 1 lb pkgs.	
	8	Northwoods Cranb	3	2	12 - 12 oz jars	
	9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	
	10	Ikura	4	8	12 - 200 ml jars	
	11	Queso Cabrales	5	4	1 kg pkg.	
	12	Queso Manchego	5	4	10 - 500 g pkgs.	
	13	Konbu	6	8	2 kg box	
	14	Tofu	6	7	40 - 100 g pkgs.	23
	15	Genen Shouyu	6	2	24 - 250 ml bottles	1
	16	Pavlova	7	3	32 - 500 g boxes	17
	17	Alice Mutton	7	6	20 - 1 kg tins	
	18	Camarvon Tigers	7	8	16 kg pkg.	6

次のコードは、WinForms FlexGrid で列方向の検索を可能にします。

C#

```
// 一番上の行からの列単位の検索を有効にします
c1FlexGrid1.AutoSearch = C1.Win.C1FlexGrid.AutoSearchEnum.FromTop;
c1FlexGrid1.AutoSearchDelay = 2;
```

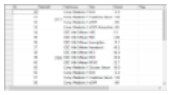
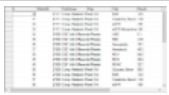
VB.NET

```
' 一番上の行からの列単位の検索を有効にします
c1FlexGrid1.AutoSearch = C1.Win.C1FlexGrid.AutoSearchEnum.FromTop
c1FlexGrid1.AutoSearchDelay = 2
```

結合

FlexGrid を使用すると、同じ値を持つセルを結合して、それらを複数の行または列にまたがって表示できます。この機能は、グリッドに表示されるデータの見栄えとわかりやすさを向上させるために役立ちます。これらの設定の効果は、HTML <ROWSPAN> および <COLSPAN> タグに似ています。

セルの結合には、いくつかの使用方法があります。たとえば、セルの結合を使用して、結合テーブルヘッダー、結合データビュー、テキストが隣接する列にはみ出るグリッドなどを作成できます。このセクションでは、以下のトピックでこれらのシナリオについて説明します。

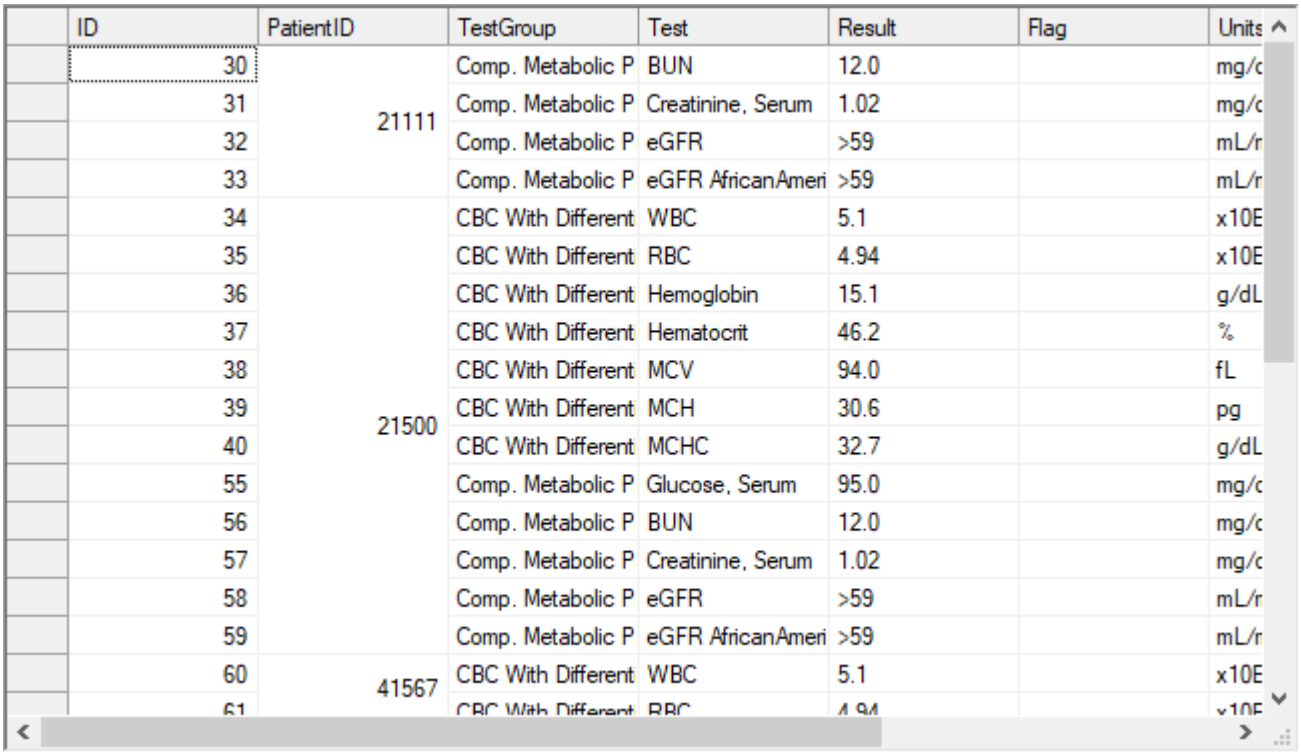
トピック	スナップショット	コンテンツ
自動結合		グリッドで自動結合を可能にする方法について説明します。 • 無制限自動結合 • 制限付き自動結合 • テーブルヘッダーの結合
はみ出して表示されるテキストの処理		通常のセルやノード行で長いテキストを表示する方法について説明します。 • 通常のセルのはみ出しの処理 • ノード行のテキストの処理
カスタム結合		デフォルトの結合動作をカスタマイズする方法について説明します。 • 方法 1：IComparer インタフェースの使用 • 方法 2：GetMergedRange メソッドのオーバーライド

自動結合

FlexGrid で、グリッド内のセルを自動結合できるようにするには、AllowMerging プロパティを **None** 以外の値に設定します。また、対象となる行または列それぞれの **AllowMerging** プロパティを **true** に設定する必要もあります。セルの結合は、隣接するセルに同じ空以外の文字列が含まれる場合にのみ行われます。1 組のセルを強制的に結合する方法はありません。結合は、セルコンテンツに基づいて自動的に行われます。この機能により、ソートされたデータで、隣接する行の値に同じデータが繰り返し表示される場合に、簡単に結合ビューを提供できます。

無制限自動結合

無制限自動結合は、隣接するセルに同じ値があるという 1 つの条件だけに基づいてセルを結合する機能です。行または列内のセルを自動的に結合するには、C1FlexGrid クラスの **AllowMerging** プロパティを **Free** に設定し、対象となる行または列オブジェクトの **AllowMerging** プロパティを **true** に設定するだけです。



	ID	PatientID	TestGroup	Test	Result	Flag	Units
	30	21111	Comp. Metabolic P	BUN	12.0		mg/c
	31		Comp. Metabolic P	Creatinine, Serum	1.02		mg/c
	32		Comp. Metabolic P	eGFR	>59		mL/r
	33		Comp. Metabolic P	eGFR AfricanAmeri	>59		mL/r
	34	21500	CBC With Different	WBC	5.1		x10E
	35		CBC With Different	RBC	4.94		x10E
	36		CBC With Different	Hemoglobin	15.1		g/dL
	37		CBC With Different	Hematocrit	46.2		%
	38		CBC With Different	MCV	94.0		fL
	39		CBC With Different	MCH	30.6		pg
	40		CBC With Different	MCHC	32.7		g/dL
	55		Comp. Metabolic P	Glucose, Serum	95.0		mg/c
	56		Comp. Metabolic P	BUN	12.0		mg/c
	57		Comp. Metabolic P	Creatinine, Serum	1.02		mg/c
	58		Comp. Metabolic P	eGFR	>59		mL/r
	59		Comp. Metabolic P	eGFR AfricanAmeri	>59		mL/r
	60	41567	CBC With Different	WBC	5.1		x10E
	61		CBC With Different	RBC	4.94		x10E

次のコードは、WinForms FlexGrid の列に無制限結合を適用する方法を示しています。

C#

```
// 許可される結合の種類を指定します
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.Free;

// 結合を許可する列を指定します
c1FlexGrid1.Cols[2].AllowMerging = true;
```

VB.NET

```
' 許可される結合の種類を指定します
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.Free
' 結合を許可する列を指定します
c1FlexGrid1.Cols(2).AllowMerging = True
```

制限付き自動結合

多くのシナリオでは、上または左方向にあるセルも結合されている場合にのみ、同じ値を持つグリッドセルを結合するようにグリッドを設定します。この動作を制限付き自動結合といい、これを行うには、**C1FlexGrid.AllowMerging** プロパティを **RestrictAll**、**RestrictRows**、または **RestrictCols** に設定します。さらに、対象となる行または列の **AllowMerging** プロパティを **true** に設定する必要があります。

WinForms FlexGrid の列で制限付き自動結合を実行するには、次のコードを使用します。

C#

```
// 結合を許可する列を指定します
c1FlexGrid1.Cols[3].AllowMerging = true;

// 左側のセルが結合されている場合にのみ列を結合します
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.RestrictCols;
```

VB.NET

```
' 結合を許可する列を指定します
c1FlexGrid1.Cols(3).AllowMerging = True
' 左側のセルが結合されている場合にのみ列を結合します
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.RestrictCols
```

テーブルヘッダーの結合

グリッドやテーブルのヘッダーセルの結合は、特にヘッダーが複数行である場合によく使用されます。ヘッダーセルを結合するには、**C1FlexGrid.AllowMerging** プロパティを **FixedOnly** に設定する必要があります。さらに、指定された行および列の **AllowMerging** プロパティを **true** に設定する必要があります。次の例では、同じ値を持つヘッダーセルを結合して、外観を簡略化しています。

	Company	Sales				Purchases			
		Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
	[2,1]	[2,2]	[2,3]	[2,4]	[2,5]	[2,6]	[2,7]	[2,8]	[2,9]
	[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	[3,6]	[3,7]	[3,8]	[3,9]
	[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	[4,6]	[4,7]	[4,8]	[4,9]
	[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	[5,6]	[5,7]	[5,8]	[5,9]
	[6,1]	[6,2]	[6,3]	[6,4]	[6,5]	[6,6]	[6,7]	[6,8]	[6,9]
	[7,1]	[7,2]	[7,3]	[7,4]	[7,5]	[7,6]	[7,7]	[7,8]	[7,9]
	[8,1]	[8,2]	[8,3]	[8,4]	[8,5]	[8,6]	[8,7]	[8,8]	[8,9]
	[9,1]	[9,2]	[9,3]	[9,4]	[9,5]	[9,6]	[9,7]	[9,8]	[9,9]
	[10,1]	[10,2]	[10,3]	[10,4]	[10,5]	[10,6]	[10,7]	[10,8]	[10,9]
	[11,1]	[11,2]	[11,3]	[11,4]	[11,5]	[11,6]	[11,7]	[11,8]	[11,9]
	[12,1]	[12,2]	[12,3]	[12,4]	[12,5]	[12,6]	[12,7]	[12,8]	[12,9]
	[13,1]	[13,2]	[13,3]	[13,4]	[13,5]	[13,6]	[13,7]	[13,8]	[13,9]
	[14,1]	[14,2]	[14,3]	[14,4]	[14,5]	[14,6]	[14,7]	[14,8]	[14,9]
	[15,1]	[15,2]	[15,3]	[15,4]	[15,5]	[15,6]	[15,7]	[15,8]	[15,9]
	[16,1]	[16,2]	[16,3]	[16,4]	[16,5]	[16,6]	[16,7]	[16,8]	[16,9]
	[17,1]	[17,2]	[17,3]	[17,4]	[17,5]	[17,6]	[17,7]	[17,8]	[17,9]
	[18,1]	[18,2]	[18,3]	[18,4]	[18,5]	[18,6]	[18,7]	[18,8]	[18,9]
	[19,1]	[19,2]	[19,3]	[19,4]	[19,5]	[19,6]	[19,7]	[19,8]	[19,9]

次のコードは、WinForms FlexGrid のテーブルヘッダーにのみ結合を適用する方法を示しています。

C#

```
// 固定の行と列の結合を許可します
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly;
```

VB.NET

```
' 固定の行と列の結合を許可します
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.FixedOnly
```

カスタム自動結合

FlexGrid には、**AllowMerging** 列挙を使用した **Custom** オプションも用意されています。この場合は、**C1FlexGrid** クラスの **MergedRanges** プロパティを使用して提供されるセル範囲コレクションに対して自動結合が実行されます。カスタム自動結合は、セルコンテンツとは無関係に実行されます。たとえば、次の図では、指定された 2 つのセル範囲が、セルの値が異なるにもかかわらず、結合されています。

	Lecturer	Start	Finish	Subject	Room No.
	Sam Anders	10:00	12:00	Engineering Physics	197
	Sam Anders		15:00	Engineering Physics	197-A
	Sam Anders	15:30	17:00	Quantum Physics	267
	Shraddha Punit	09:30	11:30	Computer Organization	267
				Computer Communication Network	126
	Shraddha Punit			Computer Communication Network	126-B
				Business Applications of IT	347
	James Ru	11:30	13:00	Business Applications of IT	347-C
	James Ru	14:00	15:30	Principles of Management	114
	Noah Anderson	09:30	11:30	Principles of Management	114
	Noah Anderson	13:30	15:30	Business Applications of IT	347-C
	Liam Will	13:30	15:30	Introduction to Web Technology	215
	Liam Will	16:00	17:30	Introduction to Web Technology	215-B
	William Trump	09:00	11:00	RDBMS	235
	William Trump	11:30	13:30	RDBMS	235-A
	William Trump	15:00	17:00	RDBMS	235
	Olivia Jackson	09:00	11:00	Communication Skills	417
	Olivia Jackson	11:30	13:00	Communication Skills	417-A
	Olivia Jackson	13:30	15:30	Software Project Presentation	330
	Olivia Jackson	16:00	17:30	Software Project Presentation	330-B

次のコードスニペットを使用して、WinForms FlexGrid にカスタム結合を適用できます。

C#

```
// 結合のモードをアクティブにします
this.c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.Custom;
// いくつかのマージされた範囲を定義します
// 2つのセルは幅が広く、1つのセルは高いです
this.c1FlexGrid1.MergedRanges.Add(2, 2, 2, 3);
// 3つのセルは幅が広く、3つのセルは高さです
this.c1FlexGrid1.MergedRanges.Add(5, 2, 7, 4);
```

VB.NET

```
' 結合のモードをアクティブにします
Me.c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.Custom
' いくつかのマージされた範囲を定義します
' 2つのセルは幅が広く、1つのセルは高いです
Me.c1FlexGrid1.MergedRanges.Add(2, 2, 2, 3)
' 3つのセルは幅が広く、3つのセルは高さです
Me.c1FlexGrid1.MergedRanges.Add(5, 2, 7, 4)
```

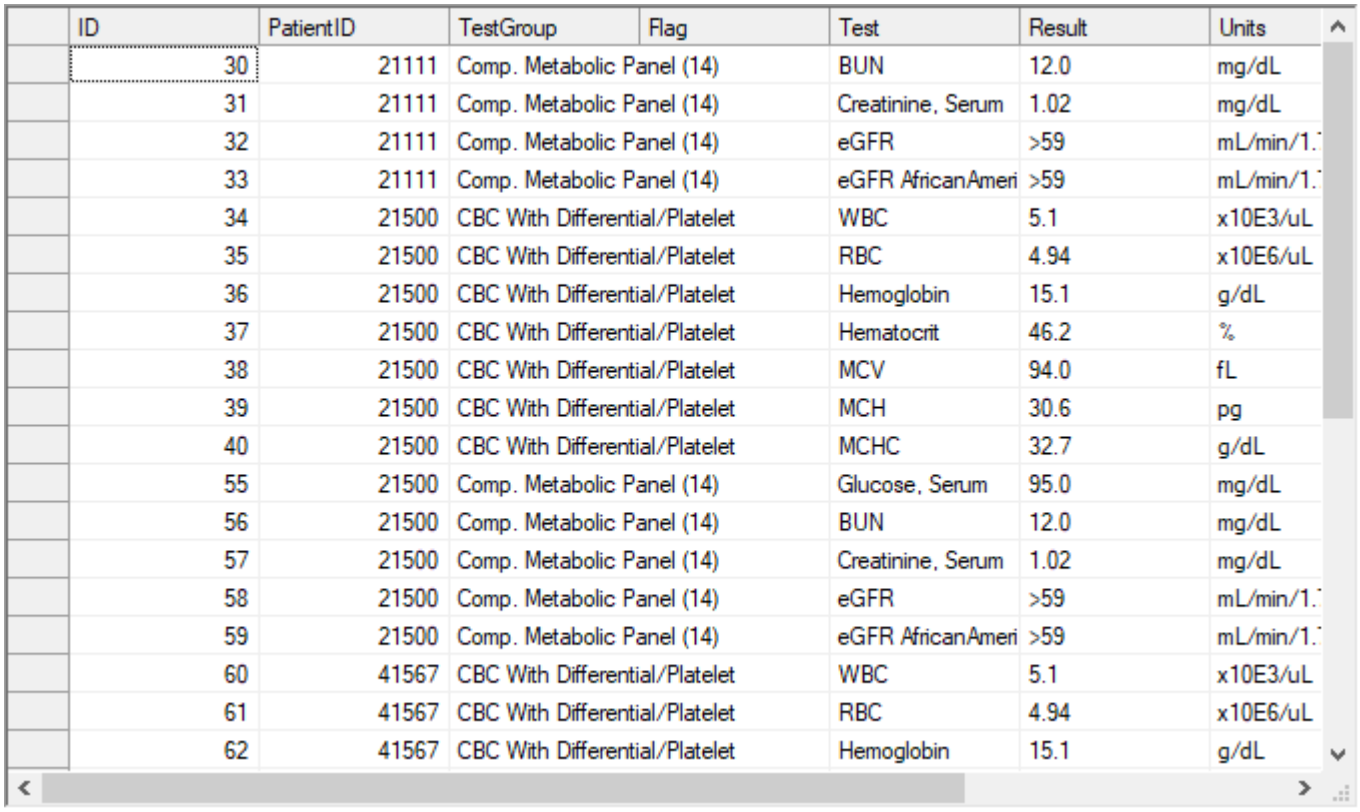
はみ出して表示されるテキストの処理

FlexGrid では、Excel と同様に、長いテキスト入力を隣接する空のセルにはみ出して表示する動作を設定できます。この動作は、通常のセルに長いテキストが入力された場合だけでなく、データ行と異なるアウトラインノードを表示する場合にも使用できます。このトピックでは、以下のセクションで、この 2 つのシナリオについて説明します。この設定は、グリッドレベルで機能するため、特定の行または列の AllowMerging プロパティを設定する必要はありません。

通常のセルのはみ出しの処理

FlexGrid のこの設定は、長すぎて 1 つのセルに収まらないテキストを隣接する空のセルにはみ出して表示します。セルに長

い入力があり、かつ隣接するセルが空の場合、入力は隣接するセルにはみ出して表示され、必要なだけのスペースを占有します。この動作を行うには、AllowMerging プロパティを **Spill** に設定します。たとえば、次の図では、長いデータ文字列が TestGroup 列のセルから空の Flag 列のセルにはみ出して表示されています。これにより、可視性が向上すると共に、グリッド内の空きスペースを活用できます。



ID	PatientID	TestGroup	Flag	Test	Result	Units
30	21111	Comp. Metabolic Panel (14)		BUN	12.0	mg/dL
31	21111	Comp. Metabolic Panel (14)		Creatinine, Serum	1.02	mg/dL
32	21111	Comp. Metabolic Panel (14)		eGFR	>59	mL/min/1.
33	21111	Comp. Metabolic Panel (14)		eGFR AfricanAmeri	>59	mL/min/1.
34	21500	CBC With Differential/Platelet		WBC	5.1	x10E3/uL
35	21500	CBC With Differential/Platelet		RBC	4.94	x10E6/uL
36	21500	CBC With Differential/Platelet		Hemoglobin	15.1	g/dL
37	21500	CBC With Differential/Platelet		Hematocrit	46.2	%
38	21500	CBC With Differential/Platelet		MCV	94.0	fL
39	21500	CBC With Differential/Platelet		MCH	30.6	pg
40	21500	CBC With Differential/Platelet		MCHC	32.7	g/dL
55	21500	Comp. Metabolic Panel (14)		Glucose, Serum	95.0	mg/dL
56	21500	Comp. Metabolic Panel (14)		BUN	12.0	mg/dL
57	21500	Comp. Metabolic Panel (14)		Creatinine, Serum	1.02	mg/dL
58	21500	Comp. Metabolic Panel (14)		eGFR	>59	mL/min/1.
59	21500	Comp. Metabolic Panel (14)		eGFR AfricanAmeri	>59	mL/min/1.
60	41567	CBC With Differential/Platelet		WBC	5.1	x10E3/uL
61	41567	CBC With Differential/Platelet		RBC	4.94	x10E6/uL
62	41567	CBC With Differential/Platelet		Hemoglobin	15.1	g/dL

次のコードは、WinForms FlexGrid で、長いテキストを隣接する空のセルにはみ出して表示できるようにします。

C#

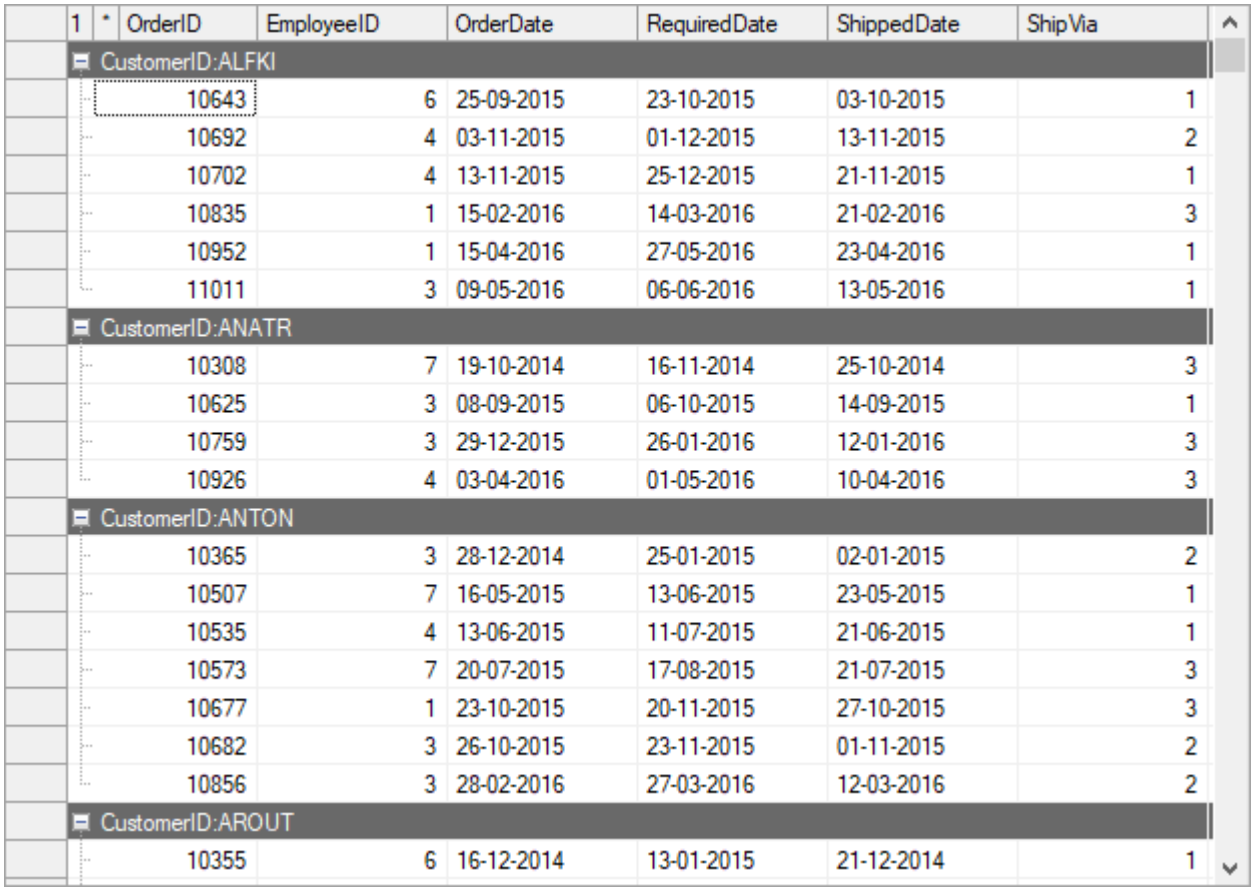
```
// 長いテキストを隣接する空のセルにこぼします
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.Spill;
```

VB.NET

```
' 長いテキストを隣接する空のセルにこぼします
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.Spill
```

ノード行のテキストの処理

Nodes 設定は、Spill に似ていますが、アウトラインノードにのみ適用されます。この設定は、データがグループに分けられており、ノード行にデータ行と異なる形式の情報が含まれている場合に便利です。それには、**C1FlexGrid.AllowMerging** プロパティを **Nodes** に設定します。



1	OrderID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia
CustomerID:ALFKI						
	10643	6	25-09-2015	23-10-2015	03-10-2015	1
	10692	4	03-11-2015	01-12-2015	13-11-2015	2
	10702	4	13-11-2015	25-12-2015	21-11-2015	1
	10835	1	15-02-2016	14-03-2016	21-02-2016	3
	10952	1	15-04-2016	27-05-2016	23-04-2016	1
	11011	3	09-05-2016	06-06-2016	13-05-2016	1
CustomerID:ANATR						
	10308	7	19-10-2014	16-11-2014	25-10-2014	3
	10625	3	08-09-2015	06-10-2015	14-09-2015	1
	10759	3	29-12-2015	26-01-2016	12-01-2016	3
	10926	4	03-04-2016	01-05-2016	10-04-2016	3
CustomerID:ANTON						
	10365	3	28-12-2014	25-01-2015	02-01-2015	2
	10507	7	16-05-2015	13-06-2015	23-05-2015	1
	10535	4	13-06-2015	11-07-2015	21-06-2015	1
	10573	7	20-07-2015	17-08-2015	21-07-2015	3
	10677	1	23-10-2015	20-11-2015	27-10-2015	3
	10682	3	26-10-2015	23-11-2015	01-11-2015	2
	10856	3	28-02-2016	27-03-2016	12-03-2016	2
CustomerID:AROUT						
	10355	6	16-12-2014	13-01-2015	21-12-2014	1

次のコードは、WinForms FlexGrid のノード行で長いテキストを処理する方法を示しています。

C#

```
// ノード行の長いテキストを隣接する空のセルにスピルします
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.Nodes;
```

VB.NET

```
' ノード行の長いテキストを隣接する空のセルにスピルします
c1FlexGrid1.AllowMerging = C1.Win.C1FlexGrid.AllowMergingEnum.Nodes
```

カスタム結合

FlexGrid では、AllowMerging プロパティを使用する多くの一般的なシナリオで、結合を実行するための十分なオプションがデフォルトで提供されています。しかし、結合オプションをカスタマイズして、グリッドの外観をさらに向上させたい場合もあります。デフォルトの結合動作は以下の 2 つの方法でカスタマイズできます。

方法 1：IComparer インタフェースの使用

デフォルトでは、null 以外の同じ値を含む隣接するセルが結合されます。この場合、文字列は大文字小文字を区別して比較され、空白は含まれます。ただし、大文字小文字を区別しないで比較を行ったり、空白をトリミングしてセルを結合することもできます。それには、**IComparer** インタフェースを実装するカスタムクラスを記述し、それを C1FlexGrid クラスの **CustomComparer** プロパティに割り当てます。

方法 2：GetMergedRange メソッドのオーバーライド

もう 1 つは、**C1FlexGrid** クラスから派生された新しいクラスを作成し、**GetMergedRange** メソッドと **GetData** メソッドをオーバーライドして独自のカスタム結合ロジックを提供する方法です。このクラスは、FlexGrid 内のデータを文脈依存で解釈し、そ

れに基づいてセルを結合します。以下の例では、これがオーバーライドメソッドによって実装されています。次の例は、WinForms FlexGrid を使用した講師のタイムテーブルでカスタム結合を使用したところです。

	Lecturer	Start	Finish	Subject	Room No.
	Sam Anders	10:00	12:00	Engineering Physics	197
		13:30	15:00		197-A
		15:30	17:00	Quantum Physics	267
	Shraddha Punit	09:30	11:30	Computer Organization	267
		12:00	13:30	Computer Communication Network	126
		14:30	16:30		126-B
	James Ru	09:00	10:30	Business Applications of IT	347
		11:30	13:00		347-C
		14:00	15:30	Principles of Management	114
	Noah Anderson	09:30	11:30	Principles of Management	114
		13:30	15:30	Business Applications of IT	347-C
	Liam Will	13:30	15:30	Introduction to Web Technology	215
		16:00	17:30		215-B
	William Trump	09:00	11:00	RDBMS	235
		11:30	13:30		235-A
		15:00	17:00		235
	Olivia Jackson	09:00	11:00	Communication Skills	417
		11:30	13:00		417-A
		13:30	15:30	Software Project Presentation	330
		16:00	17:30		330-B

C#

```
public override CellRange GetMergedRange(int row, int col, bool clip)
{
  // 結合のロジックで使用するID列のインデックスを保存します
  _colIndex = Cols.IndexOf("LecturerID");
  // 結合する時にカスタムデータを使用するようにフラグを設定します
  _doingMerge = true;
  // 基本クラスの結合のロジックを呼び出します(GetDataメソッドを使用してデータを取得します)
  CellRange cellRange = base.GetMergedRange(row, col, clip);
  // GetDataが通常どおりに動作するようにフラグをリセットします
  _doingMerge = false;
  // 結合された範囲を返します
  return cellRange;
 }
 public override Object GetData(int row, int col)
  {
    // 結合範囲を決定するためのデータを取得します。
    // ID列のコンテンツを追加して、異なるIDの行のセルがマージされないようにします
    if (_doingMerge && _colIndex > -1 && col != _colIndex)
         {
          System.Diagnostics.Debug.WriteLine($"{row},{col}");
          return base.GetDataDisplay(row, col) + base.GetDataDisplay(row,
_colIndex);
         }
    // 表示、測定、編集などするデータを取得します。
    // 基本クラスに通常どおり処理させます
    return base.GetData(row, col);
    }
```

VB.NET

```
Public Overrides Function GetMergedRange(ByVal row As Integer, ByVal col As Integer,
ByVal clip As Boolean) As CellRange
    ' 結合のロジックで使用するID列のインデックスを保存します
    _colIndex = Cols.IndexOf("LecturerID")
    ' 結合する時にカスタムデータを使用するようにフラグを設定します
    _doingMerge = True
    ' 基本クラスの結合のロジックを呼び出します(GetDataメソッドを使用してデータを取得します)
    Dim cellRange As CellRange = MyBase.GetMergedRange(row, col, clip)
    ' GetDataが通常どおりに動作するようにフラグをリセットします
    _doingMerge = False
    ' 結合された範囲を返します
    Return cellRange
End Function
Public Overrides Function GetData(ByVal row As Integer, ByVal col As Integer) As
Object
    ' 結合範囲を決定するためのデータを取得します。
    ' ID列のコンテンツを追加して、異なるIDの行のセルがマージされないようにします
    If _doingMerge AndAlso _colIndex > -1 AndAlso col <> _colIndex Then
        Debug.WriteLine($"{row},{col}")
        Return MyBase.GetDataDisplay(row, col) + MyBase.GetDataDisplay(row,
_colIndex)
    End If

    ' 表示、測定、編集などするデータを取得します
    ' 基本クラスに通常どおり処理させます
    Return MyBase.GetData(row, col)
End Function
```

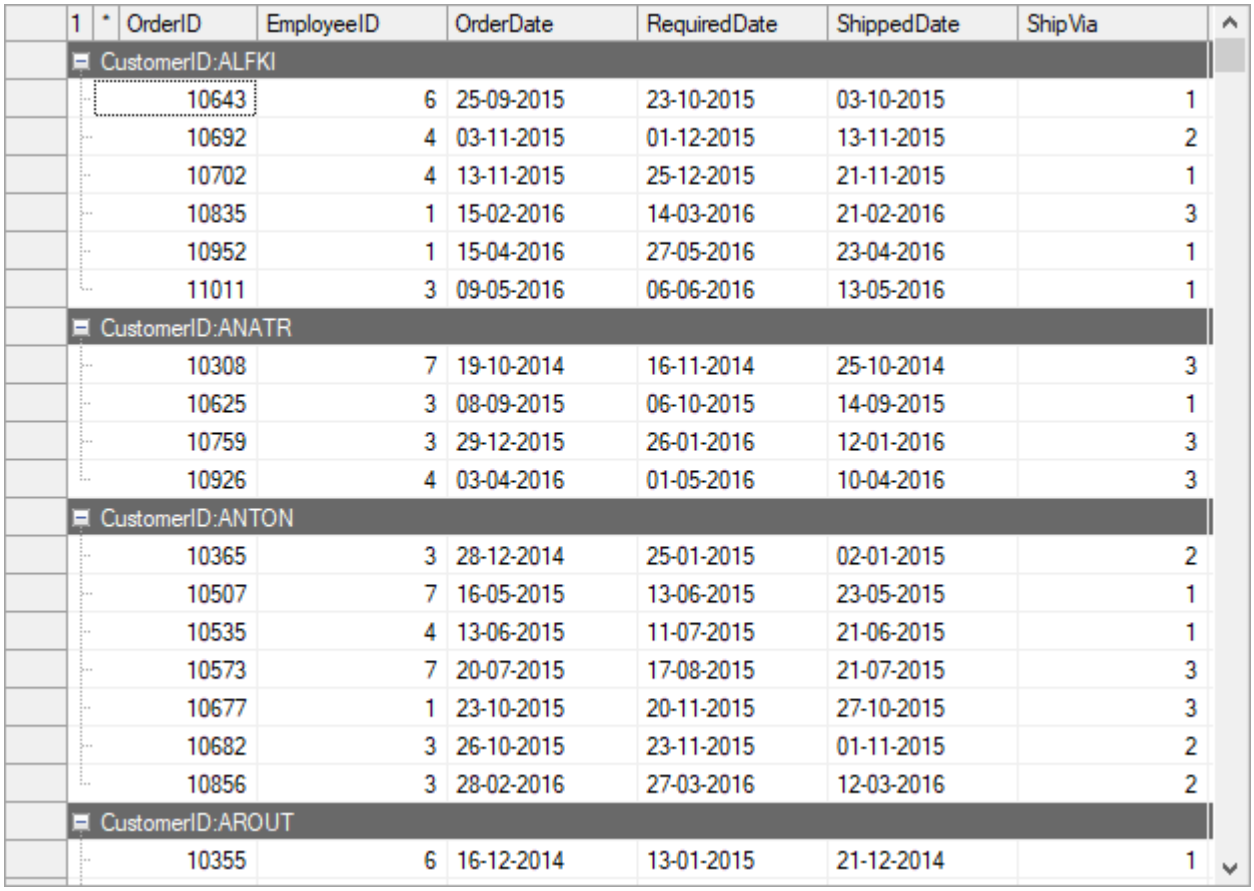
その他のカスタム結合シナリオの詳細な実装方法については、製品プロジェクト **CustomMerge**、**CustomMerge2**、**CustomMerge3**、および **CustomMerge4** を参照してください。

> **メモ**: **ComponentOneControlPanel.exe** を使用して WinForms Edition をインストールする際にサンプルをインストールした場合、前述の製品サンプルは、システムの \Documents\ComponentOne Samples\WinForms\vx.x.x\C1FlexGrid\CS にあります。

グループ

グループ化とは、共通の列値を持つ行を 1 つのグループとして表示することで、グリッドデータを階層構造に構成することです。この機能を使用すると、グループを展開したり折りたたむことで、グリッドデータの分析を容易に行うことができます。

FlexGrid のグループ化は、コードまたはユーザー操作から行うことができます。ユーザー操作の場合は、列のコンテキストメニューと **FlexGridGroupPanel** コントロールを使用します。このトピックでは、これらの方法、およびグループ化に関連するその他の操作について説明します。



1 *	OrderID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia
CustomerID:ALFKI						
	10643	6	25-09-2015	23-10-2015	03-10-2015	1
	10692	4	03-11-2015	01-12-2015	13-11-2015	2
	10702	4	13-11-2015	25-12-2015	21-11-2015	1
	10835	1	15-02-2016	14-03-2016	21-02-2016	3
	10952	1	15-04-2016	27-05-2016	23-04-2016	1
	11011	3	09-05-2016	06-06-2016	13-05-2016	1
CustomerID:ANATR						
	10308	7	19-10-2014	16-11-2014	25-10-2014	3
	10625	3	08-09-2015	06-10-2015	14-09-2015	1
	10759	3	29-12-2015	26-01-2016	12-01-2016	3
	10926	4	03-04-2016	01-05-2016	10-04-2016	3
CustomerID:ANTON						
	10365	3	28-12-2014	25-01-2015	02-01-2015	2
	10507	7	16-05-2015	13-06-2015	23-05-2015	1
	10535	4	13-06-2015	11-07-2015	21-06-2015	1
	10573	7	20-07-2015	17-08-2015	21-07-2015	3
	10677	1	23-10-2015	20-11-2015	27-10-2015	3
	10682	3	26-10-2015	23-11-2015	01-11-2015	2
	10856	3	28-02-2016	27-03-2016	12-03-2016	2
CustomerID:AROUT						
	10355	6	16-12-2014	13-01-2015	21-12-2014	1

コードによるグループ化

FlexGrid は、グリッド内のデータソース項目のグループ化方法を記述するために GroupDescriptions プロパティを提供しています。このプロパティは、**IList** インタフェース（例：List）を実装する任意のコレクションのインスタンスを値として受け取ります。コレクションの各項目は、グループ化の基準としてプロパティ名を使用してグループ化を記述します。

次のコードは、WinForms FlexGrid でコードを使用してグループ化を適用する方法を示します。

C#

```
// CustomerIDでデータをグループ化します
c1FlexGrid1.GroupDescriptions = new List<GroupDescription>() { new
GroupDescription("CustomerID") };
```

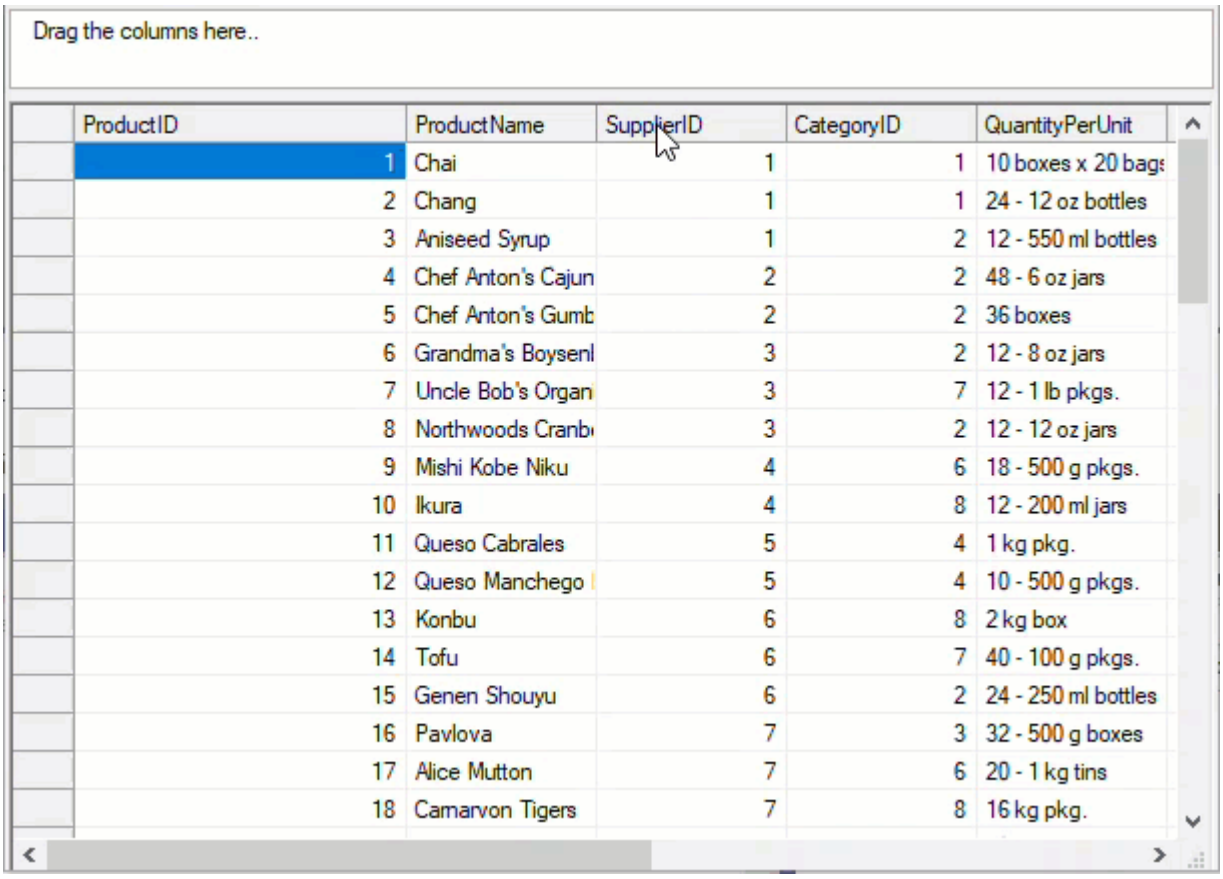
VB.NET

```
' CustomerIDでデータをグループ化します
c1FlexGrid1.GroupDescriptions = New List(Of GroupDescription)() From {
            New GroupDescription("CustomerID")
        }
```

グループパネルによるグループ化

FlexGrid では、**C1.Win.C1FlexGrid.GroupPanel** アセンブリから提供される C1FlexGridGroupPanel という拡張コントロールを使用して実行時にグループを作成することもできます。

実行時に FlexGrid でグループ化を行えるようにするには、フォームに C1FlexGridGroupPanel コントロールを追加し、それを **C1FlexGridGroupPanel** クラスの **FlexGrid** プロパティを使用して、連結先のグリッドと連結します。グリッドがグループパネルコントロールに連結されたら、列をパネルにドラッグアンドドロップして、その列でグリッドをグループ化できます。ドラッグした列をネストして階層を作成するには、複数の列を任意の順序でドラッグします。また、C1FlexGridGroupPanel クラスの **MaxGroups** プロパティを使用して、許可される最大グループ数を FlexGrid に設定することもできます。デフォルトでは、作成されたグループはすべて、展開された状態で表示されます。この設定を変更して、デフォルトでグループを折りたたまれた状態で表示するには、**AutoExpandAllGroups** プロパティを **false** に設定します。グループパネルコントロールには、どの列もパネルにドロップされていないときに表示する文字列として **Text** プロパティも用意されています。



FlexGridGroupPanel コントロールを使用して WinForms FlexGrid でグループ化を適用するには、次のコードを使用します。

C#

```
C1FlexGridGroupPanel c1FlexGridGroupPanel = new C1FlexGridGroupPanel();
c1FlexGridGroupPanel.Bounds = new System.Drawing.Rectangle(0,0, 650, 150);
this.Controls.Add(c1FlexGridGroupPanel);

// C1FlexGridGroupPanelの一般プロパティを設定します
c1FlexGridGroupPanel.FlexGrid = c1FlexGrid1;
c1FlexGridGroupPanel.Text = "Drag the columns here..";
c1FlexGridGroupPanel.MaxGroups = 5;
c1FlexGridGroupPanel.AutoExpandAllGroups = true;
```

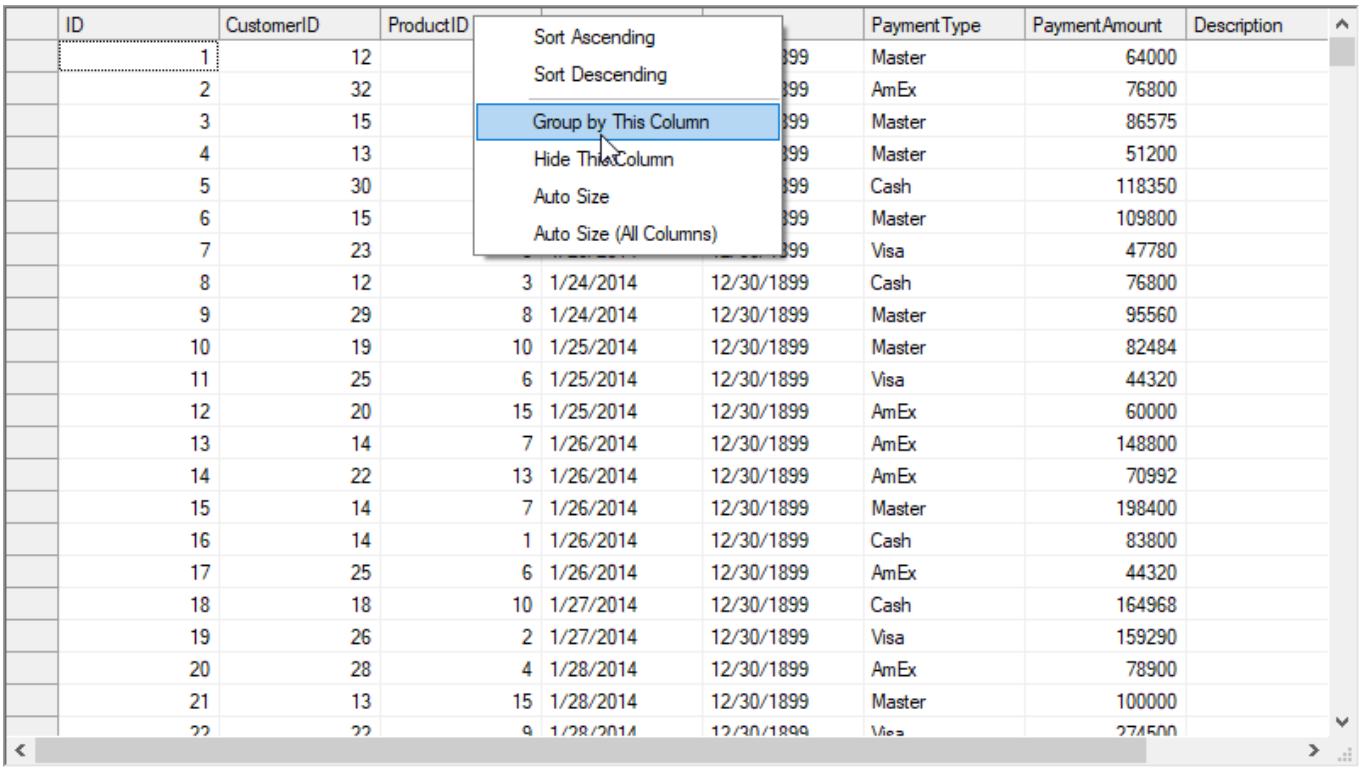
VB.NET

```
    Dim c1FlexGridGroupPanel As C1FlexGridGroupPanel = New C1FlexGridGroupPanel()
```

```
c1FlexGridGroupPanel.Bounds = New Drawing.Rectangle(0, 0, 650, 150)
Me.Controls.Add(c1FlexGridGroupPanel)

' C1FlexGridGroupPanelの一般プロパティを設定します
c1FlexGridGroupPanel.FlexGrid = c1FlexGrid1
c1FlexGridGroupPanel.Text = "Drag the columns here.."
c1FlexGridGroupPanel.MaxGroups = 5
c1FlexGridGroupPanel.AutoExpandAllGroups = True
```

コンテキストメニューによるグループ化

FlexGrid では、実行時に列操作に関連するアクションをすばやく簡単に実行するために、列コンテキストメニューがサポートされています。このコンテキストメニューのオプションの 1 つに**[この列でグループ化]**があります。このコンテキストメニューオプションを使用して、実行時に任意の列でグリッドデータをグループ化できます。グループ化が適用されると、グループ化を削除するための**[グループ化解除]**オプションがメニューに追加されます。列コンテキストメニューを使用するには、C1FlexGrid クラスにある **ColumnContextMenuEnabled** プロパティを **true** に設定する必要があります。デフォルトでは、このプロパティは false に設定されています。



ID	CustomerID	ProductID			PaymentType	PaymentAmount	Description
1	12			99	Master	64000	
2	32			99	AmEx	76800	
3	15			99	Master	86575	
4	13			99	Master	51200	
5	30			99	Cash	118350	
6	15			99	Master	109800	
7	23			99	Visa	47780	
8	12	3	1/24/2014	12/30/1899	Cash	76800	
9	29	8	1/24/2014	12/30/1899	Master	95560	
10	19	10	1/25/2014	12/30/1899	Master	82484	
11	25	6	1/25/2014	12/30/1899	Visa	44320	
12	20	15	1/25/2014	12/30/1899	AmEx	60000	
13	14	7	1/26/2014	12/30/1899	AmEx	148800	
14	22	13	1/26/2014	12/30/1899	AmEx	70992	
15	14	7	1/26/2014	12/30/1899	Master	198400	
16	14	1	1/26/2014	12/30/1899	Cash	83800	
17	25	6	1/26/2014	12/30/1899	AmEx	44320	
18	18	10	1/27/2014	12/30/1899	Cash	164968	
19	26	2	1/27/2014	12/30/1899	Visa	159290	
20	28	4	1/28/2014	12/30/1899	AmEx	78900	
21	13	15	1/28/2014	12/30/1899	Master	100000	
22	22	9	1/28/2014	12/30/1899	Visa	274500	

C#

```
// 列のコンテキストメニューを有効にします
c1FlexGrid1.ColumnContextMenuEnabled = true;
```

VB.NET

```
' 列のコンテキストメニューを有効にします
c1FlexGrid1.ColumnContextMenuEnabled = True
```

その他のグループ化オプション

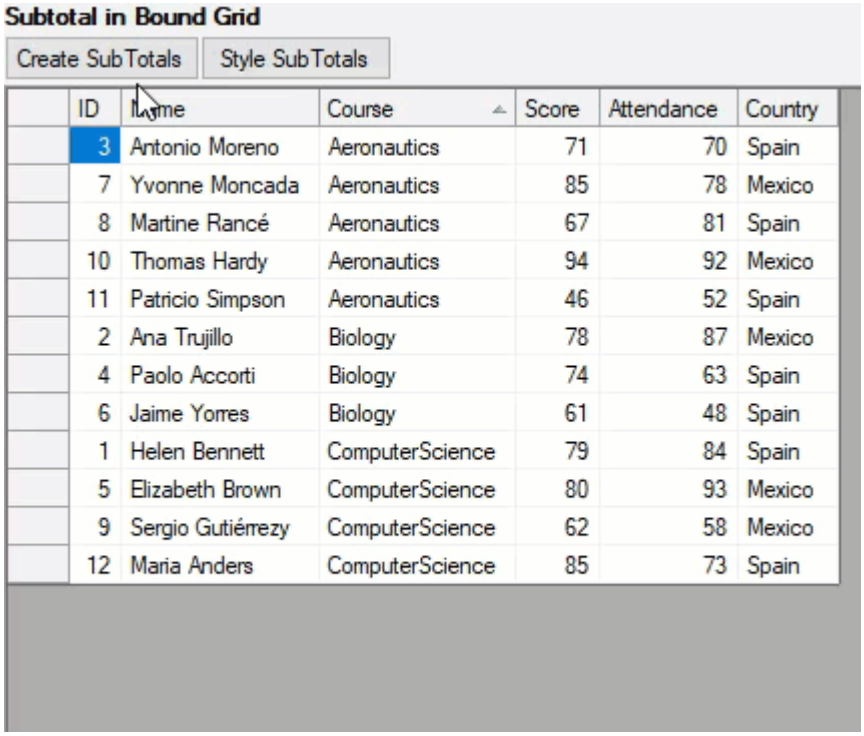
FlexGrid のグループ化には、デフォルトの機能のほかにもさまざまなオプションが用意されており、非常に柔軟に要件を満たすことができます。

- **SubtotalPosition** プロパティを使用すると、デフォルトではグループの上に表示されるグループ行の位置を変更できます。
- **HideGroupedColumns** プロパティを **true** に設定すると、グリッドのグループ化の基準となっている列を非表示にできます。
- **GroupHeaderFormat** プロパティを使用して、グループヘッダーの形式を変更できます。デフォルトでは、グループ行には、**"{name}:{value}"** の形式の文字列が表示されます。
- **C1FlexGrid.AllowMerging** プロパティを **AllowMergingEnum.Nodes** に設定すると、長いグループヘッダーのコンテンツを隣接する空のセルにはみ出して表示することができます。詳細については、「ノード行のテキストの処理」を参照してください。
- Column クラスの **Aggregate** プロパティを使用することで、列の集計値をグループヘッダーに表示できます。
- **AllowSorting** プロパティを使用すると、グループを値に基づいてソートすることができます。
- **GridTree.Style** プロパティを使用すると、FlexGrid にグループ化を表示するアウトラインツリーの外観をカスタマイズできます。詳細については、「ツリーグリッドのスタイル設定」を参照してください。

サマリー

データの集計はグリッドの最も重要な機能の 1 つです。類似したデータをグループ化して、Sum、Average、Count、Max、Min などのさまざまな集計値を簡単に計算できます。

FlexGrid では、C1FlexGrid クラスの SubTotal メソッドを使用して、簡単にデータを集計できます。このメソッドは、小計行を追加して（小計行以外の）通常の行の集計データを表示するほか、階層的な集計もサポートしています。このメソッドにはさまざまなオーバーロードがあり、集計に関連するさまざまなシナリオに対処できる柔軟性を持っています。これらのオーバーロードはいずれも AggregateEnum という共通のパラメータを持ち、実行する集計の種類を設定することができます。さまざまなオーバーロードのその他のパラメータを使用することで、アウトラインレベル、開始列と終了列、集計する値が含まれる列などを設定することができます。



小計の作成

次の例は、WinForms FlexGrid で **SubTotal** メソッドを使用して、**Score** 列に基づいて平均を計算する方法を示しています。

C#

```
// グリッドに存在する既存の小計をすべてクリアします
c1FlexGrid1.Subtotal(AggregateEnum.Clear);

// レベル:1、グループ:「Course」列、集計（平均）:Score、Attendanceでグリッドに小計を追加します
c1FlexGrid1.Subtotal(AggregateEnum.Average, 1, 3, 3, 4, "Average for {0}");
```

VB.NET

```
' グリッドに存在する既存の小計をすべてクリアします
c1FlexGrid1.Subtotal(AggregateEnum.Clear)
' レベル:1、グループ:「Course」列、集計（平均）:Score、Attendanceでグリッドに小計を追加します
c1FlexGrid1.Subtotal(AggregateEnum.Average, 1, 3, 3, 4, "Average for {0}")
```

小計のスタイル

Subtotal メソッドは、集計情報を含む行に追加する際に、新しい行に組み込みの小計スタイルを自動的に割り当てます。小

計行の外観をカスタマイズするには、デザイナの**スタイルエディタ**を使用してアウトラインスタイルのプロパティを変更するか、コードを使用します。次の例は、WinForms FlexGrid で 3 つのレベルの小計のスタイルを設定する方法を示しています。

C#

```
// 小計をレベル0でスタイル設定します
CellStyle s = c1FlexGrid1.Styles[CellStyleEnum.Subtotal0];
s.BackColor = Color.Black;
s.ForeColor = Color.White;
s.Font = new Font(c1FlexGrid1.Font, FontStyle.Bold);

// 小計をレベル1でスタイル設定します
s = c1FlexGrid1.Styles[CellStyleEnum.Subtotal1];
s.BackColor = Color.DarkBlue;
s.ForeColor = Color.White;
// 小計をレベル2でスタイル設定します
s = c1FlexGrid1.Styles[CellStyleEnum.Subtotal2];
s.BackColor = Color.DarkRed;
s.ForeColor = Color.White;
c1FlexGrid1.AutoSizeCols();
```

VB.NET

```
Dim s As CellStyle = c1FlexGrid1.Styles(CellStyleEnum.Subtotal0)
    s.BackColor = Color.Black
    s.ForeColor = Color.White
    s.Font = New Font(c1FlexGrid1.Font, FontStyle.Bold)

    s = c1FlexGrid1.Styles(CellStyleEnum.Subtotal1)
    s.BackColor = Color.DarkBlue
    s.ForeColor = Color.White

    s = c1FlexGrid1.Styles(CellStyleEnum.Subtotal2)
    s.BackColor = Color.DarkRed
    s.ForeColor = Color.White
    c1FlexGrid1.AutoSizeCols()
```

小計のカスタマイズ

FlexGrid では、組み込みの集計オプションに加えて、カスタム式を連結モードで作成し、それらをグループの小計として集計値と一緒に使用することができます。列のカスタム式は、Column クラスの **GroupExpression** プロパティを使用して指定します。次の例は、WinForms FlexGrid の **Qualified** 列の小計としてカスタム式を作成する方法を示しています。

ID	Name	Course	Score	Attendance	Country	Qualified(>5000)
Course: ComputerScience (4 items)			76.5	77		True
1	Helen Bennett	ComputerScience	79	84	Spain	6636
5	Elizabeth Brown	ComputerScience	80	93	Mexico	7440
9	Sergio Gutiérrezy	ComputerScience	62	58	Mexico	3596
12	Maria Anders	ComputerScience	85	73	Spain	6205
Course: Biology (3 items)			71	66		False
2	Ana Trujillo	Biology	78	87	Mexico	6786
4	Paolo Accorti	Biology	74	63	Spain	4662
6	Jaime Yorres	Biology	61	48	Spain	2928
Course: Aeronautics (5 items)			72.6	74.6		True
3	Antonio Moreno	Aeronautics	71	70	Spain	4970
7	Yvonne Moncada	Aeronautics	85	78	Mexico	6630
8	Martine Rancé	Aeronautics	67	81	Spain	5427
10	Thomas Hardy	Aeronautics	94	92	Mexico	8648
11	Patricio Simpson	Aeronautics	46	52	Spain	2392

C#

```
public void CustomizeSubTotal(C1FlexGrid c1FlexGrid1)
{
  List<GroupDescription> grps = new List<GroupDescription>();
  //列「Course」のグループを作成します
  GroupDescription grp = new GroupDescription("Course",
ListSortDirection.Descending, true);
  grps.Add(grp);

  //上記のグループに従ってFlexGridをグループ化します
  c1FlexGrid1.GroupDescriptions = grps;

  //カスタムの計算式を使用してFlexGridに新しい列を追加します
  var column = c1FlexGrid1.Cols.Add();
  column.Name = "Qualified";
  column.DataType = typeof(object);
  column.Caption = "Qualified(>5000)";
  column.AllowEditing = false;
  column.Expression = "[Attendance] * [Score]";

  //GroupExpression の実装
  c1FlexGrid1.Cols["Score"].GroupExpression = "=Average([Score])";
  c1FlexGrid1.Cols["Attendance"].GroupExpression = "=Average([Attendance])";
  c1FlexGrid1.Cols["Qualified"].GroupExpression = "=iif(Average([Score]*
[Attendance])<5000,false,true)";
  c1FlexGrid1.AutoSizeCols();
}
```

VB.NET

```
Public Sub CustomizeSubTotal(ByVal c1FlexGrid1 As C1FlexGrid)
    Dim grps As List(Of GroupDescription) = New List(Of GroupDescription)()
    '列「Course」のグループを作成します
    Dim grp As GroupDescription = New GroupDescription("Course",
ListSortDirection.Descending, True)
    grps.Add(grp)
    '上記のグループに従ってFlexGridをグループ化します
    c1FlexGrid1.GroupDescriptions = grps
    'カスタムの計算式を使用してFlexGridに新しい列を追加します
    Dim column = c1FlexGrid1.Cols.Add()
```

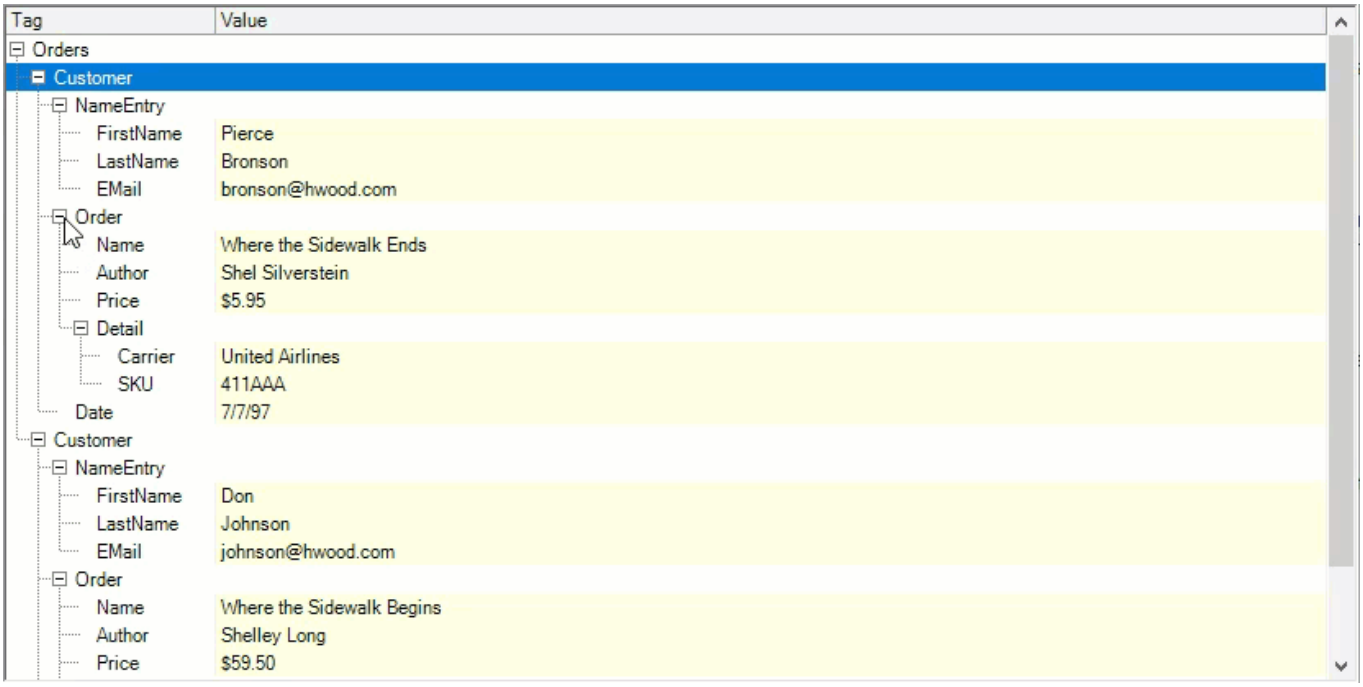
```
    column.Name = "Qualified"
    column.DataType = GetType(Object)
    column.Caption = "Qualified(>5000)"
    column.AllowEditing = False
    column.Expression = "[Attendance] * [Score]"
    'GroupExpression の実装
    c1FlexGrid1.Cols("Score").GroupExpression = "=Average([Score])"
    c1FlexGrid1.Cols("Attendance").GroupExpression = "=Average([Attendance])"
    c1FlexGrid1.Cols("Qualified").GroupExpression = "=iif(Average([Score]*
[Attendance])<5000,false,true)"
    c1FlexGrid1.AutoSizeCols()
End Sub
```

ツリーグリッド

概要

FlexGrid コントロールでよく使用される独自の機能の 1 つとして、通常のデータグリッドに階層的なグループ化を追加して、ツリーグリッドというツリー形式の構造を表示することができます。ツリーグリッドコントロールの外観は、TreeView コントロールとよく似ています。各ノード行の横に折りたたみ/展開アイコンがインデント形式の構造として表示されます。ユーザーは、ノードをクリックすることでアウトラインを展開したり折りたたんで、任意の詳細レベルを表示できます。

FlexGrid のグループ化機能を使用すれば簡単なアウトラインツリーを作成できますが、ツリーグリッドでは、顧客や注文の詳細を表示するなどのより高度なユースケースを実装できます。このようなデータを通常のグリッドで表示すると、各顧客や注文の詳細を表示することは困難です。そのような場合に、ツリーグリッドを作成してデータを階層構造にグループ化すると、情報へのアクセスと表示が容易になります。



クイック連結

ツリーグリッドへのデータのロード方法は、標準グリッドへのロード方法とまったく同じです。設計時にデータソースを使用できる場合は、Visual Studio のプロパティウィンドウを使用すると、コードを 1 行も書かずに、C1FlexGrid クラスの DataSource プロパティをグリッドに設定して、グリッドをデータに連結できます。詳細な手順については、「連結モード」を参照してください。

コードを使用して **DataSource** プロパティを設定することもできます。次のコードは、DataSource プロパティを使用して WinForms ツリーグリッドにデータを挿入する方法を示しています。

C#

```
private void Bound_Node_Load(object sender, EventArgs e)
{
  // FlexGridを連結します
   BindGrid(_gridBound);
}
public void BindGrid(C1FlexGrid grid)
{
   DataTable _dt = new DataTable();
   // データを取得します
   var fields = @" Country, City, SalesPerson, Quantity, ExtendedPrice";
   var sql = string.Format("SELECT {0} FROM Invoices ORDER BY {0}", fields);
   var da = new OleDbDataAdapter(sql, GetConnectionString());
```

```
    da.Fill(_dt);
    // グリッドにデータを連結します
    grid.DataSource = _dt;
    // ExtendedPrice 列を書式設定します
    grid.Cols["ExtendedPrice"].Format = "n2";
}
static string GetConnectionString()
{
    string path = Environment.GetFolderPath(Environment.SpecialFolder.Personal) +
@"\ComponentOne Samples\Common";
    string conn = @"provider=microsoft.jet.oledb.4.0;data source={0}\c1nwind.mdb;";
    return string.Format(conn, path);
}
```

VB.NET

```
Private Sub Bound_Node_Load(ByVal sender As Object, ByVal e As EventArgs)
        ' FlexGridを連結します
        BindGrid(_gridBound)
    End Sub
    Public Sub BindGrid(ByVal grid As C1FlexGrid)
        Dim _dt As DataTable = New DataTable()
        ' データを取得します
        Dim fields = " Country, City, SalesPerson, Quantity, ExtendedPrice"
        Dim sql = String.Format("SELECT {0} FROM Invoices ORDER BY {0}", fields)
        Dim da = New OleDbDataAdapter(sql, GetConnectionString())
        da.Fill(_dt)
        ' グリッドにデータを連結します
        grid.DataSource = _dt
        ' ExtendedPrice 列を書式設定します
        grid.Cols("ExtendedPrice").Format = "n2"
    End Sub
    Private Shared Function GetConnectionString() As String
        Dim path As String =
Environment.GetFolderPath(Environment.SpecialFolder.Personal) & "\ComponentOne
Samples\Common"
        Dim conn As String = "provider=microsoft.jet.oledb.4.0;data source=
{0}\c1nwind.mdb;"
        Return String.Format(conn, path)
    End Function
```

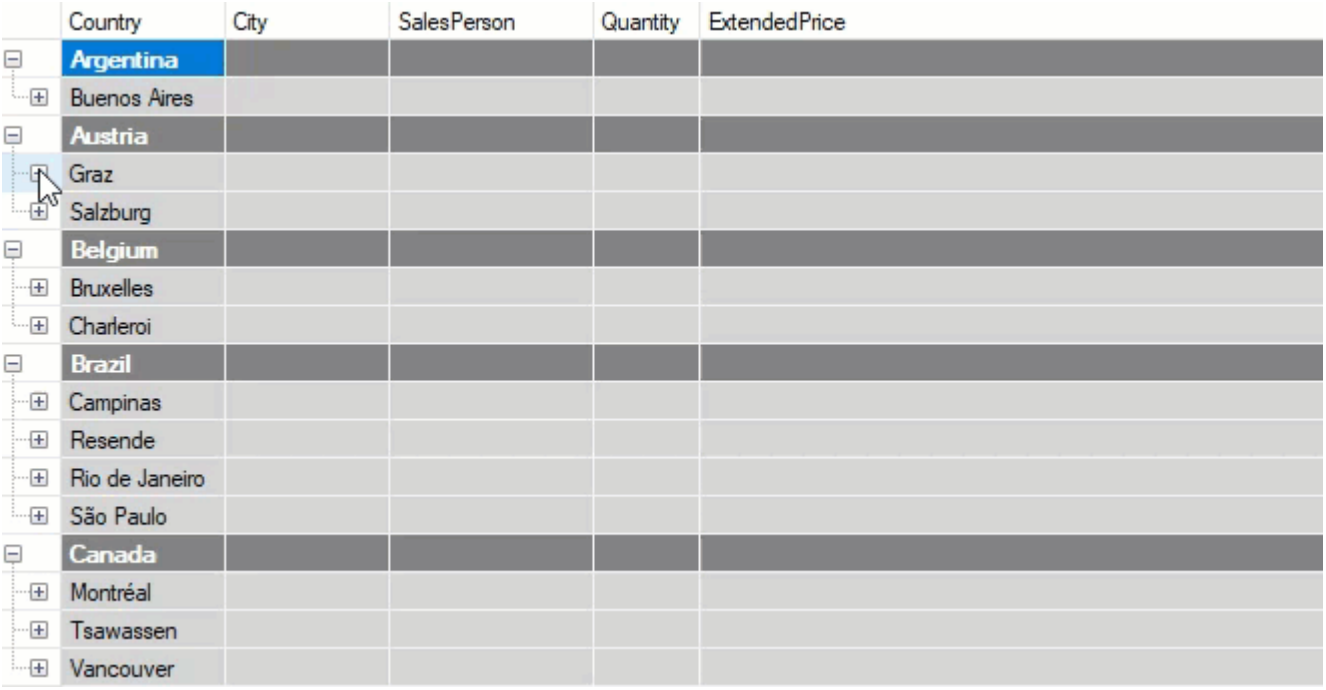
このコードでは、**OleDbDataAdapter** を使用して **DataTable** にデータを挿入し、次に DataTable をグリッドの **DataSource** プロパティに割り当てています。この標準グリッドをツリーグリッドにするには、次のセクションで説明するノード行を挿入する必要があります。

ノードの作成（連結および非連結モード）

FlexGrid では、ツリーグリッドを作成するために、ノード行の概念が導入されています。ノード行は通常のデータを含まない単なるヘッダー行ですが、ノード行の下に、通常の TreeView コントロールのノードとまったく同様に、類似するデータがグループ化されます。Level プロパティを設定することで、ノードの階層を定義することもできます。ノードを折りたたんだり展開して、ノードに含まれるデータの表示/非表示を切り替えることができます。 ツリーグリッドは、GridTree.Column プロパティで定義される任意の列に表示できます。デフォルトでは、このプロパティは -1 に設定され、ツリーは何も表示されません。

連結モード（InsertNode メソッドを使用）

ノード行は、RowCollection クラスの **InsertNode** メソッドを使用して作成できます。このメソッドは、新しいノード行を、指定されたインデックスに挿入します。これは、合計を挿入し、アウトラインを構築するための「低レベル」な方法です。



ここで使用されている **GroupBy** メソッドは、同じ値をグループ化することでノード行を挿入します。**Node** オブジェクトを取得するには、**InsertNode** メソッドの戻り値を使用するか、IsNode プロパティを使用して既存の行のノードを取得します。

次のコードを使用すると、連結 WinForms ツリーグリッドで InsertNode メソッドを使用してノードを作成できます。

C#

```
private void Bound_Node_Load(object sender, EventArgs e)
{
    // FlexGridを取得します
    BindGrid(_gridBound);
    // 連結されたFlexGridにツリーを表示します
    CreateTree(_gridBound);
    GroupBy(_gridBound, "Country", 0);
    GroupBy(_gridBound, "City", 1);
}
public void CreateTree(C1FlexGrid grid)
{
    grid.Tree.Column = 0;
    grid.Tree.Style = TreeStyleFlags.SimpleLeaf;
    grid.Tree.Show(1);
}
void GroupBy(C1FlexGrid grid, string columnName, int level)
{
    // レベル0でグループのスタイル設定します
    CellStyle s0 = grid.Styles.Add("Group0");
    s0.BackColor = Color.Gray;
    s0.ForeColor = Color.White;
    s0.Font = new Font(grid.Font, FontStyle.Bold);
    // レベル1でグループのスタイル設定します
    CellStyle s1 = grid.Styles.Add("Group1");
    s1.BackColor = Color.LightGray;
    s1.ForeColor = Color.Black;

  object current = null;
  for (int r = grid.Rows.Fixed; r < grid.Rows.Count; r++)
  {
    if (!grid.Rows[r].IsNode)
    {
```

```
      var value = grid[r, columnName];
      if (!object.Equals(value, current))
      {
        // 値が変更されました。 ノードを挿入します。
        grid.Rows.InsertNode(r, level);
        if (level == 0)
        grid.Rows[r].Style = s0;
        else if (level == 1)
        grid.Rows[r].Style = s1;
        // 最初のスクロール可能な列にグループ名を表示します
        grid[r, grid.Cols.Fixed] = value;
        // 現在の値を更新します
        current = value;
       }
      }
    }
    grid.AutoSizeCols();
}
```

VB.NET

```
Private Sub GroupBy(ByVal grid As C1FlexGrid, ByVal columnName As String, ByVal
level As Integer)
    ' レベル0でグループのスタイル設定します
    Dim s0 As CellStyle = grid.Styles.Add("Group0")
    s0.BackColor = Color.Gray
    s0.ForeColor = Color.White
    s0.Font = New Font(grid.Font, FontStyle.Bold)
    ' レベル1でグループのスタイル設定します
    Dim s1 As CellStyle = grid.Styles.Add("Group1")
    s1.BackColor = Color.LightGray
    s1.ForeColor = Color.Black
    Dim current As Object = Nothing
    Dim r As Integer = grid.Rows.Fixed

    For r = grid.Rows.Fixed To grid.Rows.Count
        If Not grid.Rows(r).IsNode Then
            Dim value = grid(r, columnName)
            If Not Object.Equals(value, current) Then
                ' 値が変更されました。 ノードを挿入します。
                grid.Rows.InsertNode(r, level)

                If level = 0 Then
                    grid.Rows(r).Style = s0
                ElseIf level = 1 Then
                    grid.Rows(r).Style = s1
                End If

                ' 最初のスクロール可能な列にグループ名を表示します
                grid(r, grid.Cols.Fixed) = value
                ' 現在の値を更新します
                current = value
            End If
        End If
    Next r

        grid.AutoSizeCols()
    End Sub
```

さらに、このコードでは、**AutoSizeCols** メソッドを呼び出して、ツリーグリッドが収まる列幅を確保しています。最後に、**GridTree.Show** メソッドを呼び出して、ノードを表示します。

また、Node クラスでは、TreeView オブジェクトモデルに基づいて次のメソッドとプロパティが提供されており、これらを使用し

てツリーグリッドを管理できます。

- **Checked**: **Node.Row** および **GridTree.Column**で定義されるセルのチェック状態を取得または設定します。
- **Collapsed/Expanded**: ノードの折りたたみ/展開状態を取得または設定します。
- **Data**: **Node.Row** と **GridTree.Column** で定義されるセル内の値を取得または設定します。
- **Image**: **Node.Row** と **GridTree.Column** で定義されるセル内の画像を取得または設定します。
- **Level**: ツリーグリッド内でノードレベルを取得または設定します。

次のプロパティおよびメソッドを使用してアウトライン構造を調べることもできます。

- **Children**: このノードの下の子ノードの数を取得します。
- **GetCellRange**: このノードに属する行の範囲を記述した **CellRange** オブジェクトを取得します。
- **GetNode**: このノードと特定の関係（親、最初の子、次の兄弟など）を持つノードを取得します。
- **Nodes**: このノードの子ノードを含むノード配列を取得します。

連結モード（Subtotal メソッドを使用）

連結モードでノードを作成するもう 1 つの方法は、Subtotal メソッドを使用することです。真に便利なツリーグリッドにするには、ノード行に含まれているデータのサマリー情報をノード行に含める必要があります。

Subtotal メソッドを使用してツリーグリッドを作成すると、小計が自動的に追加されます。このメソッドは、グリッド全体をスキャンして、グリッドデータが変化した位置にノード行とオプションの小計を自動的に挿入します。

これは、合計を挿入し、アウトラインを構築するための「高レベル」な方法です。

ID	Name	Course	Score	Attendance	Country
Average for Aeronautics			72.6		
3	Antonio Moreno	Aeronautics	71	70	Spain
7	Yvonne Moncada	Aeronautics	85	78	Mexico
8	Martine Rancé	Aeronautics	67	81	Spain
10	Thomas Hardy	Aeronautics	94	92	Mexico
11	Patricio Simpson	Aeronautics	46	52	Spain
Average for Biology			71		
2	Ana Trujillo	Biology	78	87	Mexico
4	Paolo Accorti	Biology	74	63	Spain
6	Jaime Yorres	Biology	61	48	Spain
Average for ComputerScience			76.5		
1	Helen Bennett	ComputerScience	79	84	Spain
5	Elizabeth Brown	ComputerScience	80	93	Mexico
9	Sergio Gutiérrezy	ComputerScience	62	58	Mexico
12	Maria Anders	ComputerScience	85	73	Spain

Subtotal メソッドの最初のパラメータは、さまざまな集計値（Sum、Average、Count、Max、Min など）を計算する **AggregateEnum** 列挙です。次のコードでは、**C1FlexGrid** クラスの **Subtotal** メソッドを使用して、連結 WinForms ツリーグリッドでノードを作成しています。

C#

```
private void SubtotalNode_Bound_Load(object sender, EventArgs e)
{
  // FlexGridを取得します
   BindGrid(_gridBound);
  // shows Tree in Bound FlexGrid
   CreateTree(_gridBound);
  // 連結されたFlexGridに小計を作成します
   CreateSubTotal(_gridBound);
}
```

```
public void BindGrid(C1FlexGrid grid)
{
    DataTable dt = new DataTable();
    dt.Columns.Add("ID", typeof(int));
    dt.Columns.Add("Name", typeof(string));
    dt.Columns.Add("Course", typeof(string));
    dt.Columns.Add("Score", typeof(int));
    dt.Columns.Add("Attendance", typeof(int));
    dt.Columns.Add("Country", typeof(string));

    //サンプルデータ
    dt.Rows.Add(1, "Helen Bennett", "ComputerScience", 79, 84, "Spain");
    dt.Rows.Add(2, "Ana Trujillo", "Biology", 78, 87, "Mexico");
    dt.Rows.Add(3, "Antonio Moreno", "Aeronautics", 71, 70, "Spain");
    dt.Rows.Add(4, "Paolo Accorti", "Biology", 74, 63, "Spain");
    dt.Rows.Add(5, "Elizabeth Brown", "ComputerScience", 80, 93, "Mexico");
    dt.Rows.Add(6, "Jaime Yorres", "Biology", 61, 48, "Spain");
    dt.Rows.Add(7, "Yvonne Moncada", "Aeronautics", 85, 78, "Mexico");
    dt.Rows.Add(8, "Martine Rance", "Aeronautics", 67, 81, "Spain");
    dt.Rows.Add(9, "Sergio Gutierrezy", "ComputerScience", 62, 58, "Mexico");
    dt.Rows.Add(10, "Thomas Hardy", "Aeronautics", 94, 92, "Mexico");
    dt.Rows.Add(11, "Patricio Simpson", "Aeronautics", 46, 52, "Spain");
    dt.Rows.Add(12, "Maria Anders", "ComputerScience", 85, 73, "Spain");
    grid.DataSource = dt;
    grid.AutoSizeCols();
    (grid.DataSource as DataTable).DefaultView.Sort = "Course";
}

// FlexGridでツリーを作成します
public void CreateTree(C1FlexGrid grid)
{
    grid.Tree.Column = 1;
    grid.Tree.Style = TreeStyleFlags.SimpleLeaf;
    grid.Tree.Show(1);
    grid.AutoSizeCols();
}
public void CreateSubTotal(C1FlexGrid grid)
{
    // グリッドに存在する既存の小計をすべてクリアします
    grid.Subtotal(AggregateEnum.Clear);
    // レベル:1、グループ:「Course」列、集計(平均):Score、Attendanceでグリッドに小計を追加します
    grid.Subtotal(AggregateEnum.Average, 1, 3, 3, 4, "Average for {0}");
    grid.AutoSizeCols();
}
```

VB.NET

```
Private Sub SubtotalNode_Bound_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' FlexGridを取得します
    BindGrid(_gridBound)
' 連結されたFlexGridにツリーを表示します
    CreateTree(_gridBound)
    ' Creates Subtotal(s) in Bound FlexGrid
    CreateSubTotal(_gridBound)
End Sub
Public Sub BindGrid(ByVal grid As C1FlexGrid)
    Dim dt As DataTable = New DataTable()
    dt.Columns.Add("ID", GetType(Integer))
    dt.Columns.Add("Name", GetType(String))
    dt.Columns.Add("Course", GetType(String))
    dt.Columns.Add("Score", GetType(Integer))
    dt.Columns.Add("Attendance", GetType(Integer))
    dt.Columns.Add("Country", GetType(String))
```

```
    'サンプルデータ
    dt.Rows.Add(1, "Helen Bennett", "ComputerScience", 79, 84, "Spain")
    dt.Rows.Add(2, "Ana Trujillo", "Biology", 78, 87, "Mexico")
    dt.Rows.Add(3, "Antonio Moreno", "Aeronautics", 71, 70, "Spain")
    dt.Rows.Add(4, "Paolo Accorti", "Biology", 74, 63, "Spain")
    dt.Rows.Add(5, "Elizabeth Brown", "ComputerScience", 80, 93, "Mexico")
    dt.Rows.Add(6, "Jaime Yorres", "Biology", 61, 48, "Spain")
    dt.Rows.Add(7, "Yvonne Moncada", "Aeronautics", 85, 78, "Mexico")
    dt.Rows.Add(8, "Martine Rance", "Aeronautics", 67, 81, "Spain")
    dt.Rows.Add(9, "Sergio Gutierrezy", "ComputerScience", 62, 58, "Mexico")
    dt.Rows.Add(10, "Thomas Hardy", "Aeronautics", 94, 92, "Mexico")
    dt.Rows.Add(11, "Patricio Simpson", "Aeronautics", 46, 52, "Spain")
    dt.Rows.Add(12, "Maria Anders", "ComputerScience", 85, 73, "Spain")
    grid.DataSource = dt
    grid.AutoSizeCols()
    TryCast(grid.DataSource, DataTable).DefaultView.Sort = "Course"
End Sub

' FlexGridでツリーを作成します
Public Sub CreateTree(ByVal grid As C1FlexGrid)
    grid.Tree.Column = 1
    grid.Tree.Style = TreeStyleFlags.SimpleLeaf
    grid.Tree.Show(1)
    grid.AutoSizeCols()
End Sub
Public Sub CreateSubTotal(ByVal grid As C1FlexGrid)
    ' グリッドに存在する既存の小計をすべてクリアします
    grid.Subtotal(AggregateEnum.Clear)
    ' レベル:1、グループ:「Course」列、集計(平均):Score、Attendanceでグリッドに小計を追加します
    grid.Subtotal(AggregateEnum.Average, 1, 3, 3, 4, "Average for {0}")
    grid.AutoSizeCols()
End Sub
```

非連結モード(Subtotal メソッドを使用)

Subtotal メソッドは、ツリーグリッドを柔軟に作成できるたいへん便利な方法です。このメソッドには多くのオーバーロードがあり、これらを使用して、どの列をグループ化して合計を計算するかをインデックスまたは名前で指定したり、挿入するノード行にキャプションを含めるかどうか、グループ化をどのように実行するかなどを指定することができます。

	Direction	Region	Rnd	Rnd	Rnd	Rnd	Rnd	Rnd	Rnd
	Grand Total		5,512	4,608	5,218	5,596	7,074	7,630	6,346
	Total for North		622	1,009	1,120	315	1,030	1,785	255
1	Inbound	North	298	17	515	180	832	886	40
2	Inbound	North	324	992	605	135	198	899	215
	Total for South		2,042	1,581	1,811	2,156	2,125	1,825	2,133
3	Inbound	South	462	140	334	417	771	477	422
4	Inbound	South	286	942	178	641	61	533	820
5	Inbound	South	721	410	602	966	971	221	845
6	Inbound	South	573	89	697	132	322	594	46
	Total for East		1,005	477	1,415	1,957	1,890	1,504	1,496
7	Outbound	East	774	81	430	514	703	446	881
8	Outbound	East	156	60	447	672	835	838	12
9	Outbound	East	75	336	538	771	352	220	603
	Total for West		1,843	1,541	872	1,168	2,029	2,516	2,462
10	Outbound	West	582	2	168	227	677	370	888

次のコードでは、**C1FlexGrid** クラスの Subtotal メソッドを使用して、非連結 WinForms ツリーグリッドでノードを作成しています。

C#

```
private void Unbound_Subtotal_Load(object sender, EventArgs e)
 {
   // 連結されていないFlexGridにデータを追加します
   PopulateGrid(_gridUnbound);
   // 連結されていないFlexGridにツリーを表示します
   ShowTreeNode(_gridUnbound);
   // 連結されていないFlexGridに小計を作成します
   CreateSubTotal(_gridUnbound);
 }
public void PopulateGrid(C1FlexGrid grid)
{
   // グリッドにデータを入力します
   Random rnd = new Random();
   grid.Rows.Count = 14;
   grid[0, 1] = "Direction";
   grid[0, 2] = "Region";
   CellRange rg = grid.GetCellRange(0, 3, 0, grid.Cols.Count - 1);
   rg.Data = "Rnd";
   for (int r = 1; r < grid.Rows.Count; r++)
   {
     grid[r, 0] = r;
     grid[r, 1] = (r < 7) ? "Inbound" : "Outbound";
     grid[r, 2] = (r < 3) ? "North" : (r < 7) ? "South" : (r < 10) ? "East" :
"West";
     for (int c = 3; c < grid.Cols.Count; c++)
     {
       grid[r, c] = rnd.Next(1000);
       grid.Cols[c].Format = "#,###";
     }
    }
    grid.AutoSizeCols();
}
// FlexGridでツリーを作成します
private void ShowTreeNode(C1FlexGrid grid)
{
  // ツリーを作成します
  grid.Tree.Column = 1;
  grid.Tree.Style = TreeStyleFlags.SimpleLeaf;
  grid.AutoSizeCols();
}
// Creates Subtotal in FlexGrid
public void CreateSubTotal(C1FlexGrid grid)
{
   // Clears any existing subtotal(s) present in grid
   grid.Subtotal(AggregateEnum.Clear);
   for (int c = 3; c < grid.Cols.Count; c++)
   {
     // Adds subtotals in grid
     grid.Subtotal(AggregateEnum.Sum, 0, -1, c, "Grand Total");
     grid.Subtotal(AggregateEnum.Sum, 2, 2, c, "Total for {0}");
    }
    grid.AutoSizeCols();
}
```

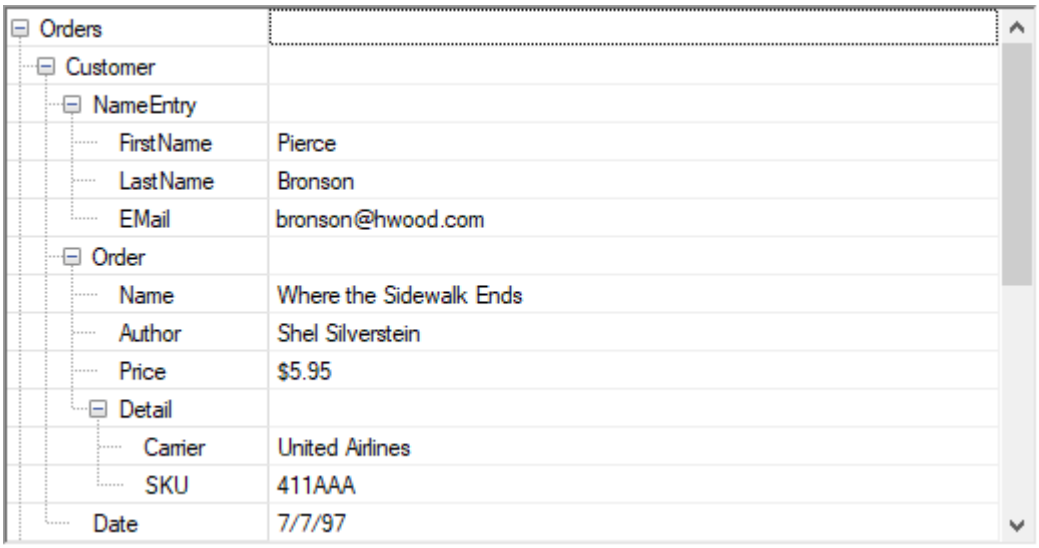
VB.NET

```
Private Sub Unbound_Subtotal_Load(ByVal sender As Object, ByVal e As EventArgs)
        ' 連結されていないFlexGridにデータを追加します
        PopulateGrid(_gridUnbound)
        ' 連結されていないFlexGridにツリーを表示します
```

```
        ShowTreeNode(_gridUnbound)
        ' 連結されていないFlexGridに小計を作成します
        CreateSubTotal(_gridUnbound)
End Sub
Public Sub PopulateGrid(ByVal grid As C1FlexGrid)
    ' グリッドにデータを入力します
    Dim rnd As Random = New Random()
    grid.Rows.Count = 14
    grid(0, 1) = "Direction"
    grid(0, 2) = "Region"
    Dim rg As CellRange = grid.GetCellRange(0, 3, 0, grid.Cols.Count - 1)
    rg.Data = "Rnd"
    For r = 1 To grid.Rows.Count - 1
        grid(r, 0) = r
        grid(r, 1) = If(r < 7, "Inbound", "Outbound")
        grid(r, 2) = If(r < 3, "North", If(r < 7, "South", If(r < 10, "East",
"West")))
        For c = 3 To grid.Cols.Count - 1
            grid(r, c) = rnd.[Next](1000)
            grid.Cols(c).Format = "#,###"
        Next
    Next
    grid.AutoSizeCols()End Sub' FlexGridでツリーを作成します
Private Sub ShowTreeNode(ByVal grid As C1FlexGrid)
    ' ツリーを作成しますFlexGridでツリーを作成します
    grid.Tree.Column = 1
    grid.Tree.Style = TreeStyleFlags.SimpleLeaf
    grid.AutoSizeCols()
End Sub
' FlexGridで小計を作成します
Public Sub CreateSubTotal(ByVal grid As C1FlexGrid)
    ' グリッドに存在する既存の小計をすべてクリアします
    grid.Subtotal(AggregateEnum.Clear)
    For c = 3 To grid.Cols.Count - 1
        ' グリッドに小計を追加します
        grid.Subtotal(AggregateEnum.Sum, 0, -1, c, "Grand Total")
        grid.Subtotal(AggregateEnum.Sum, 2, 2, c, "Total for {0}")    Next
    grid.AutoSizeCols()
End Sub
```

非連結モード(IsNode プロパティを使用)

非連結グリッドでは、**IsNode** プロパティを **true** に設定するだけで、通常の行をノード行にすることができます。通常のデータ連結行をノードにしようとすると、例外が生成されます。



次のコードを使用すると、非連結 WinForms ツリーグリッドで IsNode プロパティを使用してノードを作成できます。

C#

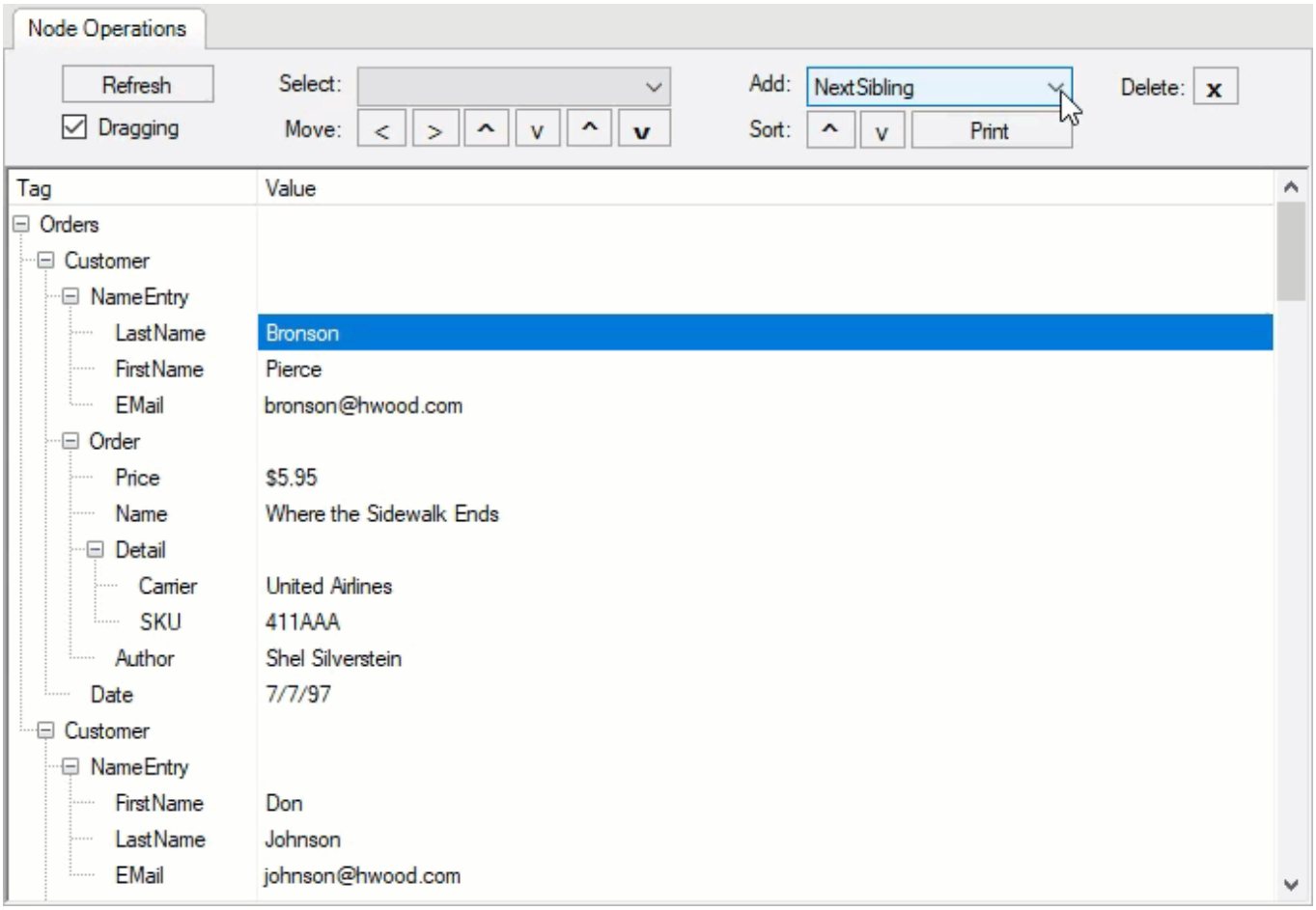
```
private void Unbound_Row_Load(object sender, EventArgs e)
{
  // 連結されていないFlexGridにデータを入力します
  PopulateGrid(_gridUnbound);
  // 連結されていないFlexGridにツリーを表示します
  ShowTreeNode(_gridUnbound);
}
private void PopulateGrid(C1FlexGrid grid)
{
  // FlexGridをリセットします
  grid.Rows.Count = 0;
  grid.Cols.Count = 2;
  string fileName = "../../test.xml";
  // xmlドキュメントをロードします
  XmlDocument xdoc = new XmlDocument();
  xdoc.Load(fileName);
  // XMLドキュメントを読み取り、FlexGridのノードとして表示します
  ShowNode(_gridUnbound, xdoc.ChildNodes[1], 0);
  grid.AutoSizeCols();
}

// FlexGridでツリーを作成します
private void ShowTreeNode(C1FlexGrid grid)
{
   // ツリーを作成します
   grid.Tree.Column = 0;
   grid.Tree.Style = TreeStyleFlags.SimpleLeaf;
   grid.AutoSizeCols();
}
// FlexGridにxmlノードを追加します
private void ShowNode(C1FlexGrid grid, XmlNode node, int level)
{
  // コメントノードをスキップします
   if (node.NodeType == XmlNodeType.Comment)
  return;
  // 読み取りxmlノードの新しい行を追加します
  int row = grid.Rows.Count;
  grid.Rows.Add();
  // グリッドのセルにxmlノードデータを割り当てます
  grid[row, 0] = node.Name;
  // xmlノードに子が1つしかないかどうかを確認します
   if (node.ChildNodes.Count == 1)
   {
     // グリッドのセルにノード値を設定します
     grid[row, 1] = node.InnerText;
   }
    // 新しい行をノードにします
     grid.Rows[row].IsNode = true;
     grid.Rows[row].Node.Level = level;
    // ノードに子がある場合は、それらも取得します
     if (node.ChildNodes.Count > 1)
     {
       // 子を取得するために再帰します
       foreach (XmlNode child in node.ChildNodes)
       ShowNode(_gridUnbound, child, level + 1);
     }
}
```

VB.NET

```
Private Sub Unbound_Row_Load(ByVal sender As Object, ByVal e As EventArgs)
        ' 連結されていないFlexGridにデータを入力します
        PopulateGrid(_gridUnbound)
        ' 連結されていないFlexGridにツリーを表示します
        ShowTreeNode(_gridUnbound)
    End Sub
    Private Sub PopulateGrid(ByVal grid As C1FlexGrid)
        ' FlexGridをリセットします
        grid.Rows.Count = 0
        grid.Cols.Count = 2
        Dim fileName As String = "../../test.xml"
        ' xmlドキュメントをロードします
        Dim xdoc As XmlDocument = New XmlDocument()
        xdoc.Load(fileName)
        ' XMLドキュメントを読み取り、FlexGridのノードとして表示します
        ShowNode(_gridUnbound, xdoc.ChildNodes(1), 0)
        grid.AutoSizeCols()
    End Sub

    ' FlexGridでツリーを作成します
    Private Sub ShowTreeNode(ByVal grid As C1FlexGrid)
        ' ツリーを作成します
        grid.Tree.Column = 0
        grid.Tree.Style = TreeStyleFlags.SimpleLeaf
        grid.AutoSizeCols()
    End Sub

    ' FlexGridにxmlノードを追加します
    Private Sub ShowNode(ByVal grid As C1FlexGrid, ByVal node As XmlNode, ByVal
level As Integer)
    ' コメントノードをスキップします
    If node.NodeType = XmlNodeType.Comment Then Return
    ' 読み取りxmlノードの新しい行を追加します
    Dim row As Integer = grid.Rows.Count
    grid.Rows.Add()
    ' グリッドのセルにxmlノードデータを割り当てます
    grid(row, 0) = node.Name
    ' xmlノードに子が1つしかないかどうかを確認します
    If node.ChildNodes.Count = 1 Then
        '  グリッドのセルにノード値を設定します
        grid(row, 1) = node.InnerText
    End If
    ' 新しい行をノードにします
    grid.Rows(row).IsNode = True
    grid.Rows(row).Node.Level = level
    ' ノードに子がある場合は、それらも取得します
    If node.ChildNodes.Count > 1 Then
        ' 子を取得するために再帰します
        For Each child As XmlNode In node.ChildNodes
            ShowNode(_gridUnbound, child, level + 1)
        Next
    End If
End Sub
```

ノードの操作

ツリーグリッドでは、データを構造化された形式で表示できるだけでなく、ノードに対してさまざまな操作を実行できます。Node クラスで提供されているさまざまなメソッドを使用して、ノードを追加、削除、移動、取得できます。



ノードの追加

Node クラスの **AddNode** メソッドを使用して、ツリーグリッド内の特定の位置にノードを追加できます。これにより、新しいノード行がコレクションに追加されます。このメソッドは、引数として **NodeTypeEnum** 列挙を受け取ります。これを使用して、別の特定のノードとの関係を指定してノードを追加できます。

次のコードは、WinForms ツリーグリッドの特定の位置にノードを追加する方法を示しています。

C#

```
private void cmbAdd_SelectionChangeCommitted(object sender, System.EventArgs e)
{
    // 現在の行ノードを取得します
    Node nd = flex.Rows[flex.Row].Node;
    // ユーザーの要求に応じて相対を追加します
    // (could be a child or a sibling)
    nd.AddNode((NodeTypeEnum)(cmbAdd.SelectedIndex + 2), cmbAdd.Text);
    flex.Focus();
}
```

VB.NET

```
Private Sub cmbAdd_SelectionChangeCommitted(ByVal sender As Object, ByVal e As
EventArgs)
    ' 現在の行ノードを取得します
    Dim nd As Node = flex.Rows(flex.Row).Node
    ' ユーザーの要求に応じて相対を追加します
    ' (could be a child or a sibling)
    nd.AddNode(CType(cmbAdd.SelectedIndex + 2, NodeTypeEnum), cmbAdd.Text)
```

```
    flex.Focus()
End Sub
```

ノードの削除

Node クラスの **RemoveNode** メソッドを使用して、選択したノードをツリーグリッドから削除できます。

次のコード例は、WinForms ツリーグリッドからノードを削除する方法を示しています。

C#

```
private void btnDelete_Click(object sender, System.EventArgs e)
{
  // 現在のノードを取得します
  Node nd = null;
  if ( flex.Rows.Count > 0 && flex.Row >= 0 && flex.Row < flex.Rows.Count)
  {
    nd = flex.Rows[flex.Row].Node;
  }
  if (nd != null)
  {
     // FlexGridからノードを削除します
     nd.RemoveNode();
     flex.Focus();
  }
}
```

VB.NET

```
Private Sub btnDelete_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' 現在のノードを取得します
    Dim nd As Node = Nothing
    If flex.Rows.Count > 0 AndAlso flex.Row >= 0 AndAlso flex.Row < flex.Rows.Count
Then
        nd = flex.Rows(flex.Row).Node
    End If
    If nd IsNot Nothing Then
        ' FlexGridからノードを削除します
        nd.RemoveNode()
        flex.Focus()
    End If
End Sub
```

ノードの移動

ツリーグリッドでは、**Node** クラスの **Move** メソッドを使用して、ノード行を別の位置に移動できます。このメソッドは、引数として **NodeMoveEnum** 列挙を受け取ります。これを使用して、ノードを移動する方向を指定できます。

次のコードを使用すると、WinForms ツリーグリッドのノードを別の位置に移動できます。

C#

```
private void btnMove_Click(object sender, System.EventArgs e)
{
   // 現在の行のノードを取得します
   Node nd = flex.Rows[flex.Row].Node;
   // ユーザーが選択した動きを適用します
   // (これにより、選択したノードが移動します)
   if (sender == btnMoveOut) nd.Move(NodeMoveEnum.Out);
   else if (sender == btnMoveIn) nd.Move(NodeMoveEnum.In);
```

```
    else if (sender == btnMoveUp) nd.Move(NodeMoveEnum.Up);
    else if (sender == btnMoveDown) nd.Move(NodeMoveEnum.Down);
    else if (sender == btnMoveFirst) nd.Move(NodeMoveEnum.First);
    else if (sender == btnMoveLast) nd.Move(NodeMoveEnum.Last);
    // ノードがまだ表示されていることを確認します
    nd.EnsureVisible();
    flex.Focus();
}
```

VB.NET

```
Private Sub btnMove_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' 現在の行のノードを取得します
    Dim nd As Node = flex.Rows(flex.Row).Node
    ' ユーザーが選択した動きを適用します
    ' （これにより、選択したノードが移動します）
    If sender Is btnMoveOut Then
        nd.Move(NodeMoveEnum.Out)
    ElseIf sender Is btnMoveIn Then
        nd.Move(NodeMoveEnum.[In])
    ElseIf sender Is btnMoveUp Then
        nd.Move(NodeMoveEnum.Up)
    ElseIf sender Is btnMoveDown Then
        nd.Move(NodeMoveEnum.Down)
    ElseIf sender Is btnMoveFirst Then
        nd.Move(NodeMoveEnum.First)
    ElseIf sender Is btnMoveLast Then
        nd.Move(NodeMoveEnum.Last)
    End If
    ' ノードがまだ表示されていることを確認します
    nd.EnsureVisible()
    flex.Focus()
End Sub
```

ノードの選択

ツリーグリッドでは、**Node** クラスの **GetNode** メソッドを使用することで、さまざまな操作を実行する対象のノードを選択できます。このメソッドは、**NodeTypeEnum** 列挙をパラメータとして受け取ります。これを使用して、別の特定のノードとの関係を指定してノードを選択できます。

次のコードは、WinForms ツリーグリッドの特定のノードを取得し、それを選択状態で表示する方法を示しています。

C#

```
private void cmbSelect_SelectionChangeCommitted(object sender, System.EventArgs e)
{
    // 現在の行のノードを取得します
    Node nd = flex.Rows[flex.Row].Node;
    // ユーザーが選択したノードを取得します
    nd = nd.GetNode((NodeTypeEnum)cmbSelect.SelectedIndex);
    // 失敗した場合は、メッセージを表示します
    if (nd == null)
    {
        MessageBox.Show("Can't find " + cmbSelect.Text + " for this node.");
        return;
    }
    // ノードを選択し、それが表示されていることを確認します（スクロールして表示します）
    nd.Select();
    nd.EnsureVisible();
    flex.Focus();
}
```

VB.NET

```
Private Sub cmbSelect_SelectionChangeCommitted(ByVal sender As Object, ByVal e As
EventArgs)
    ' 現在の行のノードを取得します
    Dim nd As Node = flex.Rows(flex.Row).Node
    ' ユーザーが選択したノードを取得します
    nd = nd.GetNode(CType(cmbSelect.SelectedIndex, NodeTypeEnum))
    ' 失敗した場合は、メッセージを表示します
    If nd Is Nothing Then
        MessageBox.Show("Can't find " & cmbSelect.Text & " for this node.")
        Return
    End If
    ' ノードを選択し、それが表示されていることを確認します（スクロールして表示します）
    nd.[Select]()
    nd.EnsureVisible()
    flex.Focus()
End Sub
```

ノードの展開と折りたたみ

下のコードに示すように、Node クラスの **Collapsed** プロパティを使用すると、ツリーグリッドアプリケーション内のすべてのノードを展開/折りたたむことができます。この機能は、ノードヘッダー間をグループとして移動する必要がある場合に便利です。

次のコードは、WinForms ツリーグリッドでノードを展開/折りたたむ方法を示しています。

C#

```
foreach (Row row in flex.Rows.Cast<Row>().Where(rw => rw.IsNode == true))
{
        Node node = row.Node;
        node.Collapsed = false;
}

foreach (Row row in flex.Rows.Cast<Row>().Where(rw => rw.IsNode == true))
{
        Node node = row.Node;
        node.Collapsed = true;
}
```

VB.NET

```
    For Each row As Row In flex.Rows.Cast(Of Row)().Where(Function(rw) rw.IsNode =
True)
        Dim node As Node = row.Node
        node.Collapsed = False
    Next

    For Each row As Row In flex.Rows.Cast(Of Row)().Where(Function(rw) rw.IsNode =
True)
        Dim node As Node = row.Node
        node.Collapsed = True
    Next
```

ノードのドラッグアンドドロップ

ツリーグリッドでは、**MouseUp**、**MouseDown**、**MouseMove** の各イベントを処理することで、選択したノードを特定の位置にドラッグアンドドロップすることができます。

次のコードは、WinForms ツリーグリッドのノードをユーザーがドラッグアンドドロップできるようにします。

C#

```
private void flex_MouseDown(object sender, MouseEventArgs e)
{
    m_DragInfo.checkDrag = false;
    // 左ボタンを押すと、移動しません。ドラッグするマウスの追跡を開始します
    if (e.Button != MouseButtons.Left) return;
    if (!chkDrag.Checked || m_DragInfo.dragging) return;
    if (flex.MouseRow < flex.Rows.Fixed || flex.MouseCol != 0) return;
    // 現在の行とマウスの位置を保存します
    m_DragInfo.row = flex.Row;
    m_DragInfo.mouseDown = new Point(e.X, e.Y);
    // チェックを開始します
    m_DragInfo.checkDrag = true;
}
private void flex_MouseMove(object sender, MouseEventArgs e)
{
    // チェックしてユーザーが許容範囲を超えた場合は、ドラッグを開始します
    if (!m_DragInfo.checkDrag || e.Button != MouseButtons.Left) return;
    if (Math.Abs(e.X - m_DragInfo.mouseDown.X) +
    Math.Abs(e.Y - m_DragInfo.mouseDown.Y) <= DRAGTOL) return;
    // フラグを更新します
    m_DragInfo.dragging = true;
    // Sets cursor and highlight node
    CellStyle cs = flex.Styles["SourceNode"];
    flex.Cursor = Cursors.NoMove2D;
    flex.SetCellStyle(m_DragInfo.row, 0, cs);
    // ここにドロップできるかどうかを確認します
    Cursor c = (NoDropHere()) ? Cursors.No : Cursors.NoMove2D;
    if (c != flex.Cursor) flex.Cursor = c;
}

private bool NoDropHere()
{
    // カーソルの下の行は無効です
    if (flex.MouseRow < flex.Rows.Fixed) return true;
    // カーソルの下の列が無効です
    if (flex.MouseCol < flex.Cols.Fixed) return true;
    if (flex.GetDataDisplay(flex.Row, 0) == "SKU") return true;
    return false;
}

private void flex_MouseUp(object sender, MouseEventArgs e)
{
    // マウスが再び下がるまでチェックしないでください
    m_DragInfo.checkDrag = false;
    // Not dragging? we're done
    if (!m_DragInfo.dragging) return;
    // ドラッグを停止します
    m_DragInfo.dragging = false;
    flex.SetCellStyle(m_DragInfo.row, 0, (CellStyle)null);
    flex.Cursor = Cursors.Default;
    // ドロップが許可されているかどうかをテストします
    if (NoDropHere()) return;
    // ノードを新しい親ノードに移動します
    Node ndSrc = flex.Rows[m_DragInfo.row].Node;
    Node ndDst = flex.Rows[flex.Row].Node;
    ndSrc.Move(NodeMoveEnum.ChildOf, ndDst);
    ndSrc.Select();
}
```

```
internal struct DRAGINFO
{
  public bool dragging;    // 現在ドラッグ中
  public bool checkDrag;   // 現在、マウスをチェックしてドラッグを開始しています
  public int row;      // ドラッグされている行のインデックス
  public Point mouseDown; //マウスダウンの位置
}
```

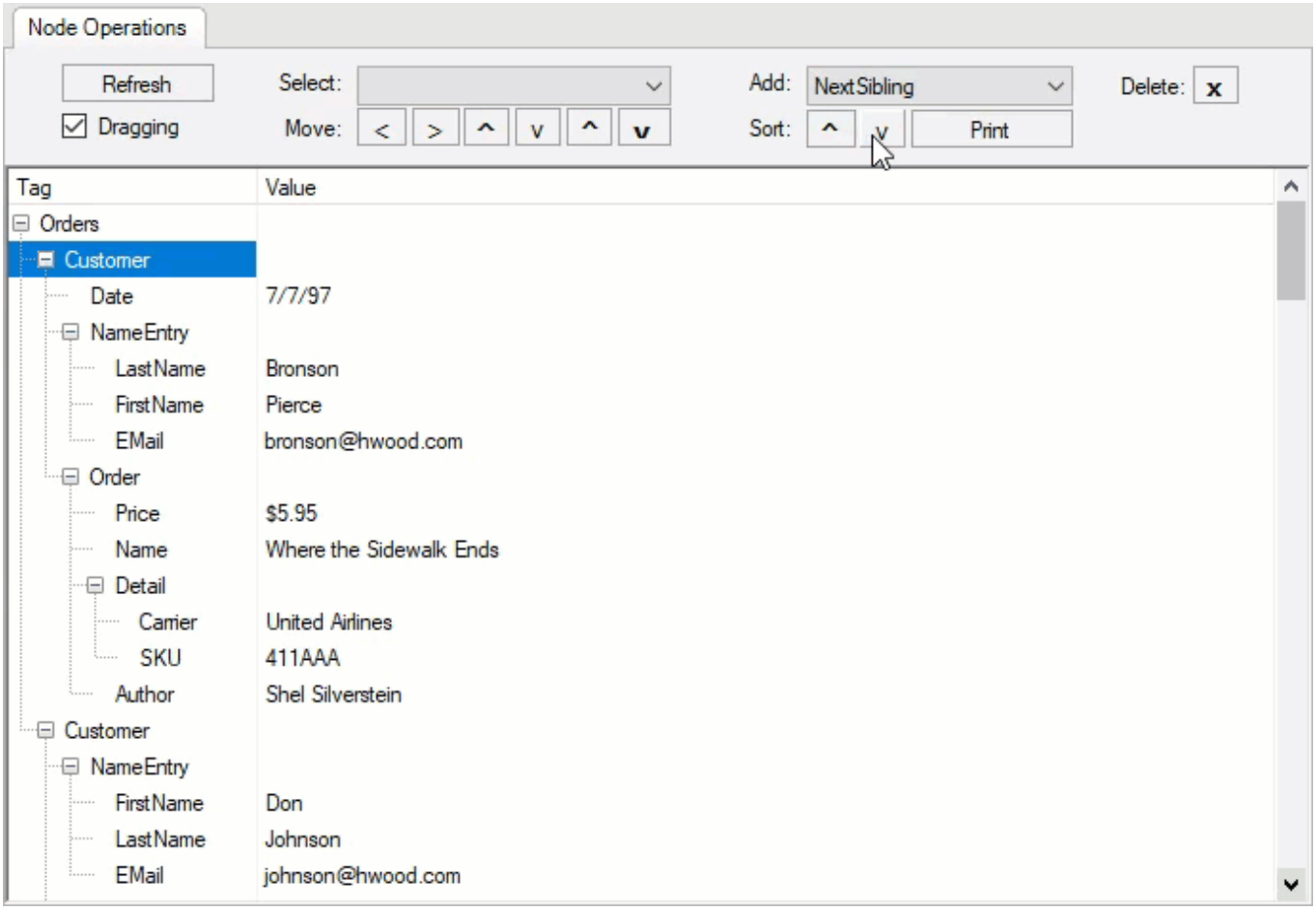
VB.NET

```
Private Sub flex_MouseDown(ByVal sender As Object, ByVal e As MouseEventArgs)
        m_DragInfo.checkDrag = False
        ' 左ボタンを押すと、移動しません。ドラッグするマウスの追跡を開始します
        If e.Button IsNot MouseButtons.Left Then Return
        If Not chkDrag.Checked OrElse m_DragInfo.dragging Then Return
        If flex.MouseRow < flex.Rows.Fixed OrElse flex.MouseCol <> 0 Then Return
        ' 現在の行とマウスの位置を保存します
        m_DragInfo.row = flex.Row
        m_DragInfo.mouseDown = New Point(e.X, e.Y)
        ' チェックを開始します
        m_DragInfo.checkDrag = True
    End Sub
    Private Sub flex_MouseMove(ByVal sender As Object, ByVal e As MouseEventArgs)
        ' チェックしてユーザーが許容範囲を超えた場合は、ドラッグを開始します
        If Not m_DragInfo.checkDrag OrElse e.Button IsNot MouseButtons.Left Then
Return
        If Math.Abs(e.X - m_DragInfo.mouseDown.X) + Math.Abs(e.Y -
m_DragInfo.mouseDown.Y) <= DRAGTOL Then Return
        ' フラグを更新します
        m_DragInfo.dragging = True
        ' カーソルを設定し、ノードを強調表示します
        Dim cs As CellStyle = flex.Styles("SourceNode")
        flex.Cursor = Cursors.NoMove2D
        flex.SetCellStyle(m_DragInfo.row, 0, cs)
        ' ここにドロップできるかどうかを確認します
        Dim c As Cursor = If((NoDropHere()), Cursors.No, Cursors.NoMove2D)
        If c IsNot flex.Cursor Then flex.Cursor = c
    End Sub
    Private Function NoDropHere() As Boolean
        ' カーソルの下の行は無効です
        If flex.MouseRow < flex.Rows.Fixed Then Return True
        ' カーソルの下の列が無効です
        If flex.MouseCol < flex.Cols.Fixed Then Return True
        If flex.GetDataDisplay(flex.Row, 0) Is "SKU" Then Return True
        Return False
    End Function
    Private Sub flex_MouseUp(ByVal sender As Object, ByVal e As MouseEventArgs)
        ' マウスが再び下がるまでチェックしないでください
        m_DragInfo.checkDrag = False
        If Not m_DragInfo.dragging Then Return
        ' ドラッグを停止します
        m_DragInfo.dragging = False
        flex.SetCellStyle(m_DragInfo.row, 0, CType(Nothing, CellStyle))
        flex.Cursor = Cursors.[Default]
        ' ドロップが許可されているかどうかをテストします
        If NoDropHere() Then Return
        ' ノードを新しい親ノードに移動します
        Dim ndSrc As Node = flex.Rows(m_DragInfo.row).Node
        Dim ndDst As Node = flex.Rows(flex.Row).Node
        ndSrc.Move(NodeMoveEnum.ChildOf, ndDst)
        ndSrc.[Select]()
    End Sub
    Friend Structure DRAGINFO
```

```
        Public dragging As Boolean    ' 現在ドラッグ中
        Public checkDrag As Boolean   ' 現在、マウスをチェックしてドラッグを開始しています
        Public row As Integer      ' ドラッグされている行のインデックス
        Public mouseDown As Point ' マウスダウンの位置
    End Structure
```

データ操作

ツリーグリッドでは、通常のグリッドと同様に、ソートや変更内容の保持など、さまざまな操作をデータレベルで実行できます。



ソート

ツリーグリッドでソートを適用するには、親ノードを選択し、**Node** クラスの **Sort** メソッドを使用して、子ノードをソートします。**SortFlags** 列挙を使用して、ソート順を指定できます。

次のコードは、WinForms ツリーグリッドでソートを適用します。

C#

```
private void btnSort_Click(object sender, System.EventArgs e)
{
  // 現在のノードを取得します
  Node nd = flex.Rows[flex.Row].Node;
  // ユーザーが選択した並べ替えを適用します
  // (これにより、選択したノードの子が並べ替えられます)
  if (sender == btnSortAscending)
   nd.Sort(SortFlags.Ascending);
 else
   nd.Sort(SortFlags.Descending);
```

```
 // 完了
  flex.Focus();
}
```

VB.NET

```
Private Sub btnSort_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' 現在のノードを取得します
    Dim nd As Node = flex.Rows(flex.Row).Node
    ' ユーザーが選択した並べ替えを適用します
    ' （これにより、選択したノードの子が並べ替えられます）
    If sender Is btnSortAscending Then
        nd.Sort(SortFlags.Ascending)
    Else
        nd.Sort(SortFlags.Descending)
    End If
    ' 完了
    flex.Focus()
End Sub
```

変更内容の保持

ここまで、高レベルの **Subtotal** メソッドと、低レベルの **InsertNode** メソッドおよび **Aggregate** メソッドを使用して、ツリーや合計を作成する方法について説明してきました。

ここで、忘れてはならないのは、ツリーグリッドはデータに基づいて作成されていますが、いかなる方法でもデータには連結されておらず、グリッドやデータが変更された場合に、それが自動的に保持されることはないということです。

たとえば、ユーザーが列の値を変更しても、小計は自動的に更新されません。また、ユーザーがグリッドをソートすると、データがリフレッシュされ、小計は表示されなくなります。

ツリーグリッドを保持するためによく使用される方法として、次の 2 つがあります。

1. ツリーを無効化するような変更をユーザーが行えないようにします。これは最も簡単なオプションです。グリッドの **AllowEditing**、**AllowDragging**、および **AllowSorting** プロパティを **false** に設定すると、ツリーに影響を与える変更をすべて禁止できます。
2. データまたはグリッドが変更されたときに、ツリーを更新します。グリッドの **AfterDataRefresh**、**AfterSort**、および **AfterEdit** イベントにハンドラをアタッチすると、アウトラインを適切に再生成できます。

通常、動的なデータ分析のための迅速でシンプルなツールを提供するには、2 番目の方法が有効です。この方法は、FlexGrid コントロールに付属する製品サンプル「**Analyze（分析）**」で説明されています。このサンプルでは、初期アウトラインを作成し、ユーザーに列の並べ替えを許可します。列の順序が変更されると、自動的にデータを再ソートし、アウトラインを再作成します。

> **メモ**: **ComponentOneControlPanel.exe** を使用して WinForms Edition をインストールする際にサンプルをインストールした場合、前述の製品サンプルは、システムの \Documents\ComponentOne Samples\WinForms\vx.x.x\C1FlexGrid\CS にあります。

「ノードの作成」セクションでは、特定の列の同じ値をグループ化してノード行を挿入する **GroupBy** メソッドの実装についても説明しました。そのコードは、呼び出されたときにいくつかのレベルのノードを追加できるように、既存のノード行をスキップしながらすべての列をスキャンし、グループ化の基準となる列の現在の値を追跡します。現在値が変化すると、ノード行が挿入され、最初のスクロール可能な列に新しいグループ名が表示されます。

ただし、このメソッドはデータがアウトライン構造に従ってソートされることを前提とするなど、グループ化を保持するにはいくつかの課題があります。また、**GroupBy** メソッドが行を挿入してグリッドがちらつく可能性があります。これを避けるには、通常は、更新を行う前に **Redraw** プロパティを **false** に設定し、更新後に **true** に戻します。

DeferRefresh クラスは、グリッドの Redraw プロパティを false に設定し、破棄されるときに元の値を復元するシンプルなユーティリティです。これにより、更新時に例外が発生した場合でも、Redraw を適切に復元できます。**BindGrid** メソッドは、アウトライン構造が必要とする順序でグリッドがソートされるようにします。

ツリーグリッドのカスタマイズ

FlexGrid では、スタイルを設定したり、ノードを展開/折りたたむチェックボックスのような要素やノードアイコンのような画像を使用することで、ツリーグリッドをカスタマイズできます。ツリーグリッドをカスタマイズすると、アウトラインツリーノードをより明確に構造化して表示できるため、見栄えがよくなると共に、読み取りやすくなります。

ツリーグリッドのスタイル設定

ツリーグリッドのスタイル設定は、FlexGrid コントロールのスタイル設定に似ています。**Tree** プロパティは、ツリーグリッドのカスタマイズに使用されるメソッドやプロパティを公開する **GridTree** オブジェクトへの参照を返します。以下は、よく使用されるプロパティのリストです。

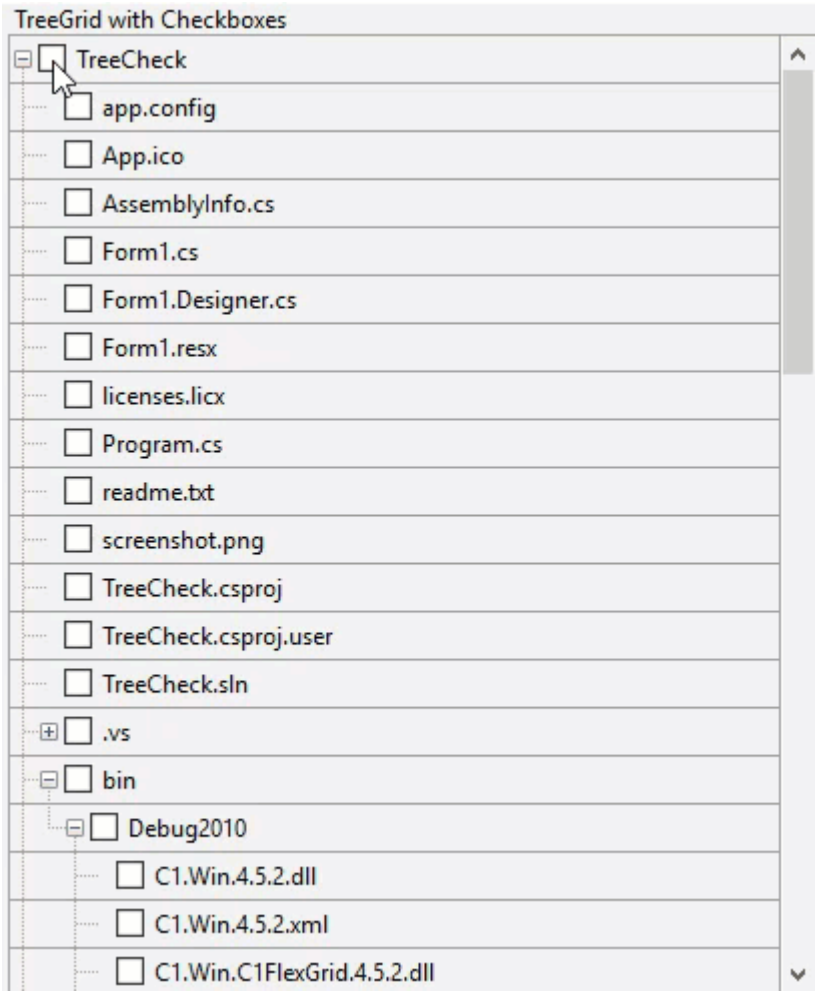
- **Column**：アウトラインツリーを含む列のインデックスを取得または設定します。このプロパティを -1 に設定すると、アウトラインツリーは非表示になります。
- **Indent**：隣接するノードレベル間のインデント（ピクセル単位）を取得または設定します。インデントレベルを大きくすると、ツリーの幅が広くなります。
- **Style**：表示するアウトラインツリーのタイプを取得または設定します。このプロパティを使用して、ユーザーがツリー全体を折りたたむ/展開するためのボタンバーをツリーの上部に置くかどうか、線やシンボルを表示するかどうか、ツリーからデータ行やノード行への接続線を表示するかどうかを決定できます。
- **LineColor**：接続線の色を取得または設定します。
- **LineStyle**：接続線のスタイルを取得または設定します。

ツリーグリッドのスタイル設定の詳細については、FlexGrid のドキュメントのトピック「スタイル設定と外観」を参照してください。

チェックボックスを持つツリーの表示

チェックボックスや画像を持つツリーグリッドを作成するには、最初に、ツリーグリッドを作成するための FlexGrid を初期化した後、**RowCollection** クラスの **AddNode** メソッドを使用してツリーにノードを追加します。

次に、ツリーグリッドでチェックボックスを実装するには、親および子ノードでチェックボックスを保持する必要があります。この方法では、**Node** クラスの **Checked** プロパティを使用して、ノードにチェックボックスを表示するかどうかを定義します。



次のコードは、WinForms ツリーグリッドのツリーノードにチェックボックスを表示する方法を示します。

C#

```
void c1FlexGrid1_CellChecked(object sender, C1.Win.C1FlexGrid.RowColEventArgs e)
 {
   // すべての子にチェック値を適用します
   var node = this.c1FlexGrid1.Rows[e.Row].Node;
   UpdateCheckChildren(node);
  // 親にチェック値を適用します
   UpdateCheckParent(node);
 }
void UpdateCheckChildren(C1.Win.C1FlexGrid.Node node)
 {
    var checkState = node.Checked;
    foreach (C1.Win.C1FlexGrid.Node child in node.Nodes)
   {
     child.Checked = checkState;
     UpdateCheckChildren(child);
    }
 }
void UpdateCheckParent(C1.Win.C1FlexGrid.Node node)
{
   // このノードの親を取得します
   var parent = node.Parent;
   if (parent != null)
   {
     // チェックされた/チェックされていない子をカウントします
     int cntChecked = 0;
```

```
     int cntUnchecked = 0;
     int cntGrayed = 0;
     foreach (C1.Win.C1FlexGrid.Node child in parent.Nodes)
    {
      switch (child.Checked)
      {
        case C1.Win.C1FlexGrid.CheckEnum.Checked:
             cntChecked++;
             break;
        case C1.Win.C1FlexGrid.CheckEnum.Unchecked:
             cntUnchecked++;
             break;
        case C1.Win.C1FlexGrid.CheckEnum.Grayed:
             cntGrayed++;
             break;
      }
     }
    // 親のチェック状態を更新します
     if (cntGrayed > 0 || (cntChecked > 0 && cntUnchecked > 0))
     {
       parent.Checked = C1.Win.C1FlexGrid.CheckEnum.Grayed;
     }
     else if (cntChecked > 0 && cntUnchecked == 0)
     {
       parent.Checked = C1.Win.C1FlexGrid.CheckEnum.Checked;
     }
     else if (cntUnchecked > 0 && cntChecked == 0)
     {
       parent.Checked = C1.Win.C1FlexGrid.CheckEnum.Unchecked;
     }
     // 祖父も更新します
       UpdateCheckParent(parent);
     }
 }
```

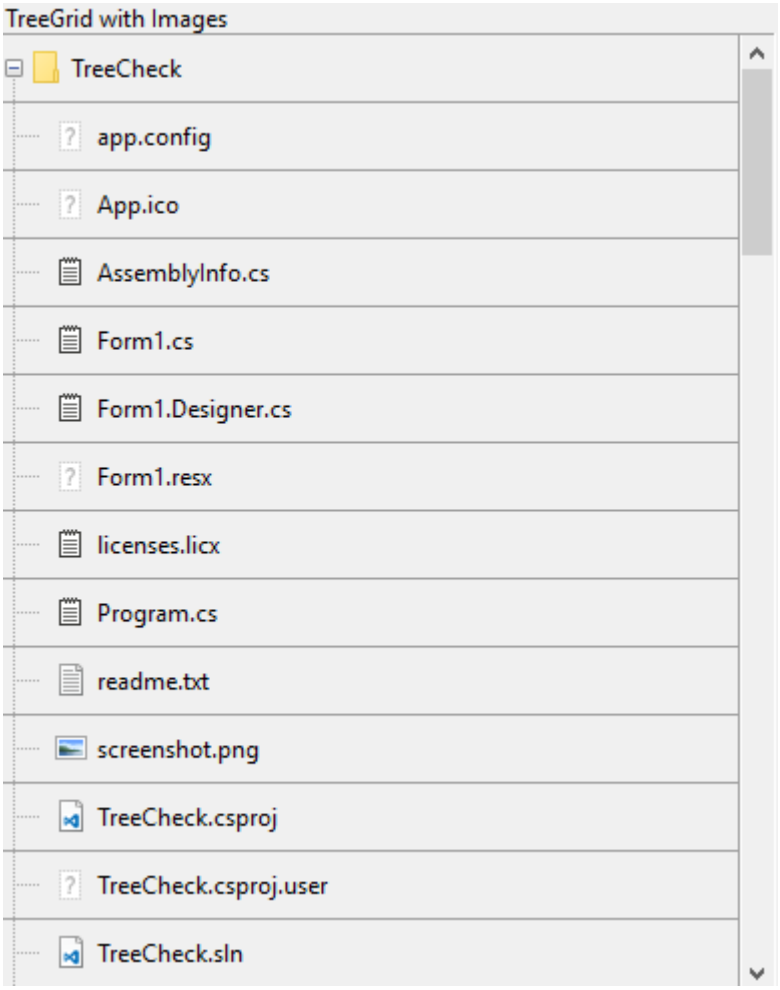
VB.NET

```
Private Sub c1FlexGrid1_CellChecked(ByVal sender As Object, ByVal e As
C1.Win.C1FlexGrid.RowColEventArgs)
        ' すべての子にチェック値を適用します
        Dim node = Me.c1FlexGrid1.Rows(e.Row).Node
        UpdateCheckChildren(node)
        ' 親にチェック値を適用します
        UpdateCheckParent(node)
    End Sub
    Private Sub UpdateCheckChildren(ByVal node As C1.Win.C1FlexGrid.Node)
        Dim checkState = node.Checked
        For Each child As C1.Win.C1FlexGrid.Node In node.Nodes
            child.Checked = checkState
            Me.UpdateCheckChildren(child)
        Next
    End Sub
    Private Sub UpdateCheckParent(ByVal node As C1.Win.C1FlexGrid.Node)
        ' このノードの親を取得します
        Dim parent = node.Parent
        If parent IsNot Nothing Then
            ' チェックされた/チェックされていない子をカウントします
            Dim cntChecked As Integer = 0
            Dim cntUnchecked As Integer = 0
            Dim cntGrayed As Integer = 0
            For Each child As C1.Win.C1FlexGrid.Node In parent.Nodes
                Select Case child.Checked
                    Case C1.Win.C1FlexGrid.CheckEnum.Checked
```

```
                    cntChecked += 1
                Case C1.Win.C1FlexGrid.CheckEnum.Unchecked
                    cntUnchecked += 1
                Case C1.Win.C1FlexGrid.CheckEnum.Grayed
                    cntGrayed += 1
            End Select
        Next
        ' 親のチェック状態を更新します
        If cntGrayed > 0 OrElse cntChecked > 0 AndAlso cntUnchecked > 0 Then
            parent.Checked = C1.Win.C1FlexGrid.CheckEnum.Grayed
        ElseIf cntChecked > 0 AndAlso cntUnchecked = 0 Then
            parent.Checked = C1.Win.C1FlexGrid.CheckEnum.Checked
        ElseIf cntUnchecked > 0 AndAlso cntChecked = 0 Then
            parent.Checked = C1.Win.C1FlexGrid.CheckEnum.Unchecked
        End If
        ' 祖父も更新します
        UpdateCheckParent(parent)
    End If
End Sub
```

画像を持つツリーの表示

ツリーグリッドのノードアイコンとして画像を追加するには、**AddImageFolder** メソッドを使用して、ファイルに関連付けられたノードを作成し、画像をノードに割り当てます。



次のコードを使用して、WinForms ツリーグリッドのノードと一緒に画像またはアイコンを表示できます。

C#

```
void AddImageFolder(string path, int level)
{
 int cnt = 0;
 try
 {
   //ディレクトリ内のすべてのファイルをループします
   foreach (string file in Directory.GetFiles(path))
   {
    //ファイルごとにノードを作成し、FlexGridに追加します
    var node = c1FlexGrid2.Rows.AddNode(level);
    //作成されたノードのデータを設定します
    node.Data = Path.GetFileName(file);

    //画像をノードに割り当てるためのコード
    ShowImage(node, file);
    //各ディレクトリから(最大で)20個のファイルのみを表示します
    cnt++;
    if (cnt > 20) break;
    }
  }
    catch { }
    try
    {
      //パスのすべてのサブディレクトリをループします
      foreach (string subPath in Directory.GetDirectories(path))
      {
       //ファイルごとにノードを作成し、FlexGridに追加します
       var node = c1FlexGrid2.Rows.AddNode(level);
       //作成されたノードのデータを設定します
       node.Data = Path.GetFileName(subPath);
       //画像をノードに割り当てるためのコード
       ShowImage(node, subPath);
       //現在のディレクトリのサブディレクトリ/ファイルをトラバースします
       AddImageFolder(subPath, level + 1);
       //各ディレクトリから(最大で)20個のファイルのみを表示します
       cnt++;
       if (cnt > 20) break;
       }
     }
      catch { }
}
```

VB.NET

```
Private Sub AddImageFolder(ByVal path As String, ByVal level As Integer)
    Dim cnt As Integer = 0
    Try
        'ディレクトリ内のすべてのファイルをループします
        For Each file As String In Directory.GetFiles(path)
            'ファイルごとにノードを作成し、FlexGridに追加します
            Dim node = c1FlexGrid2.Rows.AddNode(level)
            '作成されたノードのデータを設定します
            node.Data = Path.GetFileName(file)
            '画像をノードに割り当てるためのコード
            ShowImage(node, file)
            '各ディレクトリから(最大で)20個のファイルのみを表示します
            cnt += 1
            If cnt > 20 Then Exit For
        Next
    Catch
```

```
    End Try
    Try
        'パスのすべてのサブディレクトリをループします
        For Each subPath As String In Directory.GetDirectories(path)
            'ファイルごとにノードを作成し、FlexGridに追加します
            Dim node = c1FlexGrid2.Rows.AddNode(level)
            '作成されたノードのデータを設定します
            node.Data = Path.GetFileName(subPath)
            '画像をノードに割り当てるためのコード
            ShowImage(node, subPath)
            '現在のディレクトリのサブディレクトリ/ファイルをトラバースします
            AddImageFolder(subPath, level + 1)
            '各ディレクトリから(最大で)20個のファイルのみを表示します
            cnt += 1
            If cnt > 20 Then Exit For
        Next
    Catch
    End Try
End Sub
```

上のコードでは、**ShowImage** というカスタムメソッドを使用して、ファイル拡張子に基づいてノードの画像を設定しています。

C#

```
public void ShowImage(C1.Win.C1FlexGrid.Node node , string path)
 {
      //パスからファイル拡張子を取得します
     string extension = Path.GetExtension(path);
     //パスがファイルに属している場合
      if (extension != "")
      {
        //ファイル拡張子に基づいてノードの画像を設定します
        switch (extension)
        {
          case ".txt":
               node.Image = Properties.Resources.Txt;
               break;
           case ".resx":
               node.Image = Properties.Resources.Txt;
               break;
          case ".licx":
               node.Image = Properties.Resources.Txt;
               break;
          case ".cs":
               node.Image = Properties.Resources.Txt;
               break;
          case ".vb":
               node.Image = Properties.Resources.Txt;
               break;
          case ".exe":
               node.Image = Properties.Resources.Exe;
               break;
          case ".dll":
               node.Image = Properties.Resources.Dll;
               break;
          case ".sln":
               node.Image = Properties.Resources.VS;
               break;
          case ".csproj":
               node.Image = Properties.Resources.VS;
               break;
          case ".bmp":
```

```
                node.Image = Properties.Resources.Img;
                break;
            case ".png":
                node.Image = Properties.Resources.Img;
                break;
            case ".gif":
                node.Image = Properties.Resources.Video;
                break;
            case ".accdb":
                node.Image = Properties.Resources.Access;
                break;
            default:
                node.Image = Properties.Resources.Unknown;
                break;
            }
        }
        //それ以外のパスはディレクトリ/フォルダに属しています
        else
        {
           node.Image = Properties.Resources.Folder;
        }
 }
```

VB.NET

```
Public Sub ShowImage(ByVal node As C1.Win.C1FlexGrid.Node, ByVal path As String)
    'パスからファイル拡張子を取得します
    Dim extension As String = Path.GetExtension(path)
    'パスがファイルに属している場合
    If Not Equals(extension, "") Then
        'ファイル拡張子に基づいてノードの画像を設定します
        Select Case extension
            Case ".txt"
                node.Image = Properties.Resources.Txt
            Case ".resx"
                node.Image = Properties.Resources.Txt
            Case ".licx"
                node.Image = Properties.Resources.Txt
            Case ".cs"
                node.Image = Properties.Resources.Txt
            Case ".vb"
                node.Image = Properties.Resources.Txt
            Case ".exe"
                node.Image = Properties.Resources.Exe
            Case ".dll"
                node.Image = Properties.Resources.Dll
            Case ".sln"
                node.Image = Properties.Resources.VS
            Case ".csproj"
                node.Image = Properties.Resources.VS
            Case ".bmp"
                node.Image = Properties.Resources.Img
            Case ".png"
                node.Image = Properties.Resources.Img
            Case ".gif"
                node.Image = Properties.Resources.Video
            Case ".accdb"
                node.Image = Properties.Resources.Access
            Case Else
                node.Image = Properties.Resources.Unknown
                'それ以外のパスはディレクトリ/フォルダに属しています
        End Select
    Else
```

```
            node.Image = Properties.Resources.Folder
        End If
    End Sub
```

クリップボード

FlexGrid では、編集可能なグリッドデータに対して切り取り、コピー、貼り付けなどのさまざまなクリップボード操作を柔軟に行うことができます。一般的なクリップボードキーの自動処理を有効にするには、**AutoClipboard** プロパティを **true** に設定する必要があります。このプロパティは、以下のクリップボード操作と、それに対応するキーを処理します。

コピー	Ctrl+C、Ctrl+Ins
切り取り	Ctrl+X、Shift+Del
貼り付け	Ctrl+V、Shift+Ins
削除	Del

前述のクリップボード操作は、スタイルと画像には影響しません。グリッドヘッダーとデータに対してのみ作用します。選択したコンテンツのどの部分を行ヘッダー、列ヘッダー、およびデータからコピーするかは、**ClipboardCopyMode** プロパティを使用して指定できます。コピー範囲にデータマップまたは複数列コンボボックスが存在する場合は、表示値のみがコピーされます。また、セル範囲をコピーすると、非表示のセルもコピーされます。非表示のセルをコピーから除外する方法については、「**非表示セルの除外**」を参照してください。

以下のコードは、WinForms FlexGrid でクリップボード操作を有効にする方法を示します。

C#

```
// キーボードからクリップボード操作を有効にします
   c1FlexGrid1.AutoClipboard = true;

// データとすべてのヘッダーをコピーするようにコピーモードを設定します
   c1FlexGrid1.ClipboardCopyMode =
C1.Win.C1FlexGrid.ClipboardCopyModeEnum.DataAndAllHeaders;
```

VB.NET

```
' キーボードからクリップボード操作を有効にします
c1FlexGrid1.AutoClipboard = True

' データとすべてのヘッダーをコピーするようにコピーモードを設定します
c1FlexGrid1.ClipboardCopyMode =
C1.Win.C1FlexGrid.ClipboardCopyModeEnum.DataAndAllHeaders
```

コードによるクリップボード操作

クリップボード操作をコードで行うこともできます。次の例は、ボタンクリックでコピーおよび貼り付け操作を行う方法を示します。

WinForms FlexGrid においてコードでクリップボード操作を実行するには、次のコードを使用します。

C#

```
private void CopyButton_Click(object sender, EventArgs e)
{
 // クリップボードにコピーするグリッドのデータのみをコピーできるようにします
    c1FlexGrid1.ClipboardCopyMode = ClipboardCopyModeEnum.DataOnly;

 // Clipプロパティに改行コードを追加し、クリップボードに設定します
    Clipboard.SetDataObject(c1FlexGrid1.Clip + "\r");
    MessageBox.Show($"Copied Range: [{c1FlexGrid1.Row},{c1FlexGrid1.Col}]-
[{c1FlexGrid1.RowSel},{c1FlexGrid1.ColSel}]");
}
```

```
private void PasteButton_Click(object sender, EventArgs e)
{
 // クリップボードのテキストを取得します
    IDataObject data = Clipboard.GetDataObject();
    if (data.GetDataPresent(DataFormats.Text))
    {
        string str1, str2;
     // クリップボードからデータを取得します
        str1 = (string)data.GetData(DataFormats.Text);

     // クリップボードから最後の行のフィードコードを削除します
        str2 = str1.Remove(str1.Length - 1, 1);

     // 範囲を選択してデータを貼り付けます
        c1FlexGrid1.Select(c1FlexGrid1.Row, c1FlexGrid1.Col, c1FlexGrid1.Rows.Count
- 1, c1FlexGrid1.Cols.Count - 1, true);
        c1FlexGrid1.Clip = str2;
        c1FlexGrid1.Select(c1FlexGrid1.Row, c1FlexGrid1.Col);
    }
```

VB.NET

```
Private Sub CopyButton_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' クリップボードにコピーするグリッドのデータのみをコピーできるようにします
    c1FlexGrid1.ClipboardCopyMode = ClipboardCopyModeEnum.DataOnly

    ' Clipプロパティに改行コードを追加し、クリップボードに設定します
    Clipboard.SetDataObject(c1FlexGrid1.Clip & Microsoft.VisualBasic.Constants.vbCr)
    MessageBox.Show($"Copied Range: [{c1FlexGrid1.Row},{c1FlexGrid1.Col}]-
[{c1FlexGrid1.RowSel},{c1FlexGrid1.ColSel}]")
End Sub

Private Sub PasteButton_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' クリップボードのテキストを取得します
    Dim data As IDataObject = Clipboard.GetDataObject()

    If data.GetDataPresent(DataFormats.Text) Then
        Dim str1, str2 As String
        ' クリップボードからデータを取得します
        str1 = CStr(data.GetData(DataFormats.Text))

        ' クリップボードから最後の行のフィードコードを削除します
        str2 = str1.Remove(str1.Length - 1, 1)

        ' 範囲を選択してデータを貼り付けます
        c1FlexGrid1.Select(c1FlexGrid1.Row, c1FlexGrid1.Col, c1FlexGrid1.Rows.Count
- 1, c1FlexGrid1.Cols.Count - 1, True)
        c1FlexGrid1.Clip = str2
        c1FlexGrid1.Select(c1FlexGrid1.Row, c1FlexGrid1.Col)
    End If
```

非表示セルの除外

FlexGrid では、**AutoClipboard** プロパティを使用して有効にしたキーボード操作でセル範囲をコピーすると、デフォルトで非表示セルもコピーされます。ただし、次のコードを使用して、非表示の行や列を除外できます。

次のコードは、WinForms のツリーグリッドで行われるクリップボード操作から非表示のセルを除外する方法を示します。

C#

```
// 列を非表示にします
c1FlexGrid1.Cols[2].Visible = false;
c1FlexGrid1.AutoClipboard = true;


// 次に、Ctrl + Cを押す前に、非表示の列を含むいくつかのセルを選択します
private void c1FlexGrid1_KeyDown_1(object sender, KeyEventArgs e)
{

        // [Ctrl + C]でコピーします
        if ((e.Control == true) && (e.KeyCode == Keys.C))
        {
            // 自動処理が行われないため、キー入力を無効にします
            e.Handled = true;
            // 選択したセル範囲のCellRangeオブジェクトを取得します
            C1.Win.C1FlexGrid.CellRange cr;
            cr = c1FlexGrid1.Selection;

            string StrCopy = "";
            for (int i = cr.r1; i <= cr.r2; i++)
            {
                if (c1FlexGrid1.Rows[i].Visible == true)
                {
                    for (int j = cr.c1; j <= cr.c2; j++)
                    {
                        if (c1FlexGrid1.Cols[j].Visible == true)
                        {
                            StrCopy = StrCopy + c1FlexGrid1[i, j].ToString();
                            if (j != cr.c2)
                            {
                                StrCopy = StrCopy + "\t";
                            }
                        }
                    }
                    StrCopy = StrCopy + "\n";
                }
            }
            // クリップボードに設定します
            Clipboard.SetDataObject(StrCopy);
            MessageBox.Show("Copied data: \n" + StrCopy);
        }

}
```

VB.NET

```
    ' 列を非表示にします
    c1FlexGrid1.Cols(2).Visible = False
    c1FlexGrid1.AutoClipboard = True


' 次に、Ctrl + Cを押す前に、非表示の列を含むいくつかのセルを選択します
Private Sub c1FlexGrid1_KeyDown_1(ByVal sender As Object, ByVal e As KeyEventArgs)

    ' [Ctrl + C]でコピーします
    If e.Control = True AndAlso e.KeyCode = Keys.C Then
        ' 自動処理が行われないため、キー入力を無効にします
        e.Handled = True
        ' 選択したセル範囲のCellRangeオブジェクトを取得します
        Dim cr As C1.Win.C1FlexGrid.CellRange
        cr = c1FlexGrid1.Selection
        Dim StrCopy = ""
```

```
        For i = cr.r1 To cr.r2

            If c1FlexGrid1.Rows(i).Visible = True Then
                For j = cr.c1 To cr.c2

                    If c1FlexGrid1.Cols(j).Visible = True Then
                        StrCopy = StrCopy & c1FlexGrid1(i, j).ToString()

                        If j <> cr.c2 Then
                            StrCopy = StrCopy &
Microsoft.VisualBasic.Constants.vbTab
                        End If
                    End If
                Next

                StrCopy = StrCopy & Microsoft.VisualBasic.Constants.vbLf
            End If
        Next
        ' クリップボードに設定します
        Clipboard.SetDataObject(StrCopy)
        MessageBox.Show("Copied data: " & Microsoft.VisualBasic.Constants.vbLf &
StrCopy)
    End If
End Sub
```

保存、ロード、印刷

FlexGrid には、グリッドやそのコンテンツをテキスト、Excel、または XML ファイルに簡単に保存することができるさまざまな組み込みオプションが用意されています。さらに、保存したファイルを別のグリッドにロードしたり、グリッドの特定の状態をバックアップとして保存することもできます。また、目的の形式でグリッド画像を保存することもできます。それだけでなく、ハードコピーが必要な場合は、グリッドや、グリッド内で選択された領域を印刷することもできます。このトピックでは、グリッドで使用できる保存、ロード、および印刷関連のさまざまなオプションについて説明します。

トピック	コンテンツ
保存	グリッドとそのコンテンツを別の形式で保存する方法について説明します。 • テキストファイルとして保存 • Excel ファイルとして保存 • XML として保存 • 画像として保存
ロード	グリッドのコンテンツをさまざまなファイル形式からロードする方法について説明します。 • テキストファイルからのロード • Excel ファイルからのロード • データベースからのロード • XML からロード
印刷	グリッドの印刷方法と、印刷関連のオプションについて説明します。 • グリッドの印刷 • 印刷オプション • 印刷プレビューダイアログのカスタマイズ 

保存

FlexGrid には、テキスト、Excel、XML、画像形式などの目的の形式でグリッドを保存するためのさまざまなメソッドが用意されています。この機能は、**C1.Win.C1FlexGrid.ImportExport.dll** という名前のアセンブリを通じて有効になります。したがって、グリッドを保存するには、このアセンブリへの参照を追加する必要があります。

テキストファイルとして保存

グリッドのコンテンツをテキストファイルとして保存するには、**Extensions** クラスの **SaveGrid** メソッドを使用します。このメソッドには、区切り文字やエンコードなどを選択するパラメータや、グリッドのどの部分を保存するかを指定するパラメータがあります。生成されたテキストファイルは、後からコントロールにロードし直したり、カンマ区切りやタブ区切りファイルをサポートしている Microsoft Excel などのアプリケーションにロードし直すことができます。

次のコードは、WinForms FlexGrid のコンテンツをテキストファイルとして保存する方法を示します。

C#

```
private void btnSaveTxt_Click(object sender, EventArgs e)
{
    c1FlexGrid1.SaveGrid("../../ExportedGrid.txt", FileFormatEnum.TextComma,
FileFlags.AsDisplayed | FileFlags.IncludeFixedCells);
}
```

VB.NET

```
Private Sub btnSaveTxt_Click(ByVal sender As Object, ByVal e As EventArgs)
    c1FlexGrid1.SaveGrid("../../ExportedGrid.txt", FileFormatEnum.TextComma,
FileFlags.AsDisplayed Or FileFlags.IncludeFixedCells)
End Sub
```

Excel ファイルとして保存

グリッドを Excel ファイルとして保存するには、前述の **SaveGrid** メソッドを使用し、**FileFormatEnum.Excel** に書式パラメータを設定します。コンピュータに Microsoft Excel をインストールしておく必要はありません。ただし、SaveGrid メソッドは、1 つのワークブックの 1 つのワークシートにしかデータを保存できません。

Excel ファイルにデータを保存する際に、さらに詳細な制御を行うには、代わりに **SaveExcel** メソッドを使用します。このメソッドで Excel ファイルを保存すると、行や列のサイズ、フォント、色、書式、セルの配置など、ほとんどのデータ型と書式設定情報が変換されます。ただし、固定セル、結合セル、画像、データマップ、セル境界線などの機能は、Excel ファイルへの変換時に変換されません。

WinForms FlexGrid を Excel ファイルとして保存するには、次のコードを使用します。

C#

```
private void btnSaveExcl_Click(object sender, EventArgs e)
{
    c1FlexGrid1.SaveExcel("../../ExportedGrid.xlsx",  FileFlags.AsDisplayed |
FileFlags.IncludeFixedCells);
}
```

VB.NET

```
Private Sub btnSaveExcl_Click(ByVal sender As Object, ByVal e As EventArgs)
    c1FlexGrid1.SaveExcel("../../ExportedGrid.xlsx", FileFlags.AsDisplayed Or
FileFlags.IncludeFixedCells)
End Sub
```

XML として保存

グリッドのコンテンツを XML ドキュメントにシリアライズするには、C1FlexGrid クラスの **WriteXML** メソッドを呼び出し、XML ドキュメントのパスをパラメータとして渡します。このメソッドは、グリッドのすべてのコンテンツ（セルに格納されたデータ、行と列のプロパティ、スタイル、画像など）を XML ドキュメントにシリアライズします。独自の型のオブジェクトがグリッドに格納されている場合も、文字列との変換を提供する **System.ComponentModel.TypeConverter** を持つならシリアライズされます。

次のコードは、WinForms FlexGrid のコンテンツを XML ドキュメントに保存する方法を示しています。

C#

```
private void btnSaveXML_Click(object sender, EventArgs e)
{
    c1FlexGrid1.WriteXml("../../ExportedGrid.xml");
}
```

VB.NET

```
Private Sub btnSaveXML_Click(ByVal sender As Object, ByVal e As EventArgs)
    c1FlexGrid1.WriteXml("../../ExportedGrid.xml")
End Sub
```

画像として保存

グリッドを画像として保存するには、**C1FlexGrid** クラスの **CreateImage** メソッドを使用してグリッド画像を作成し、指定したパスにその画像オブジェクトを保存します。セル範囲を指定したり、セル範囲オブジェクトをパラメータとして渡すことで、グリッドの特定の部分を画像として保存することもできます。

WinForms FlexGrid を画像として保存するには、次のコードを使用します。

C#

```
private void btnSaveImg_Click(object sender, EventArgs e)
{
    Image gridImage = c1FlexGrid1.CreateImage();
    gridImage.Save("../../ExportedGrid.png");
}
```

VB.NET

```
Private Sub btnSaveImg_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim gridImaage As Image = c1FlexGrid1.CreateImage()
    gridImaage.Save("../../ExportedGrid.png")
End Sub
```

PDFへの保存

The easiest way of saving FlexGrid as a PDF is to use the System.PrintDocument class. You can set its PrinterName property to "**Microsoft Print to PDF**" and invoke Print method of the class.

C#

```
var printDocument = _flexGrid.PrintParameters.PrintDocument;
printDocument.DocumentName = "Export to PDF";
printDocument.PrinterSettings.PrinterName = "Microsoft Print to PDF";
// ドキュメントをMicrosoftPDFプリンターに印刷します
 printDocument.Print();
```

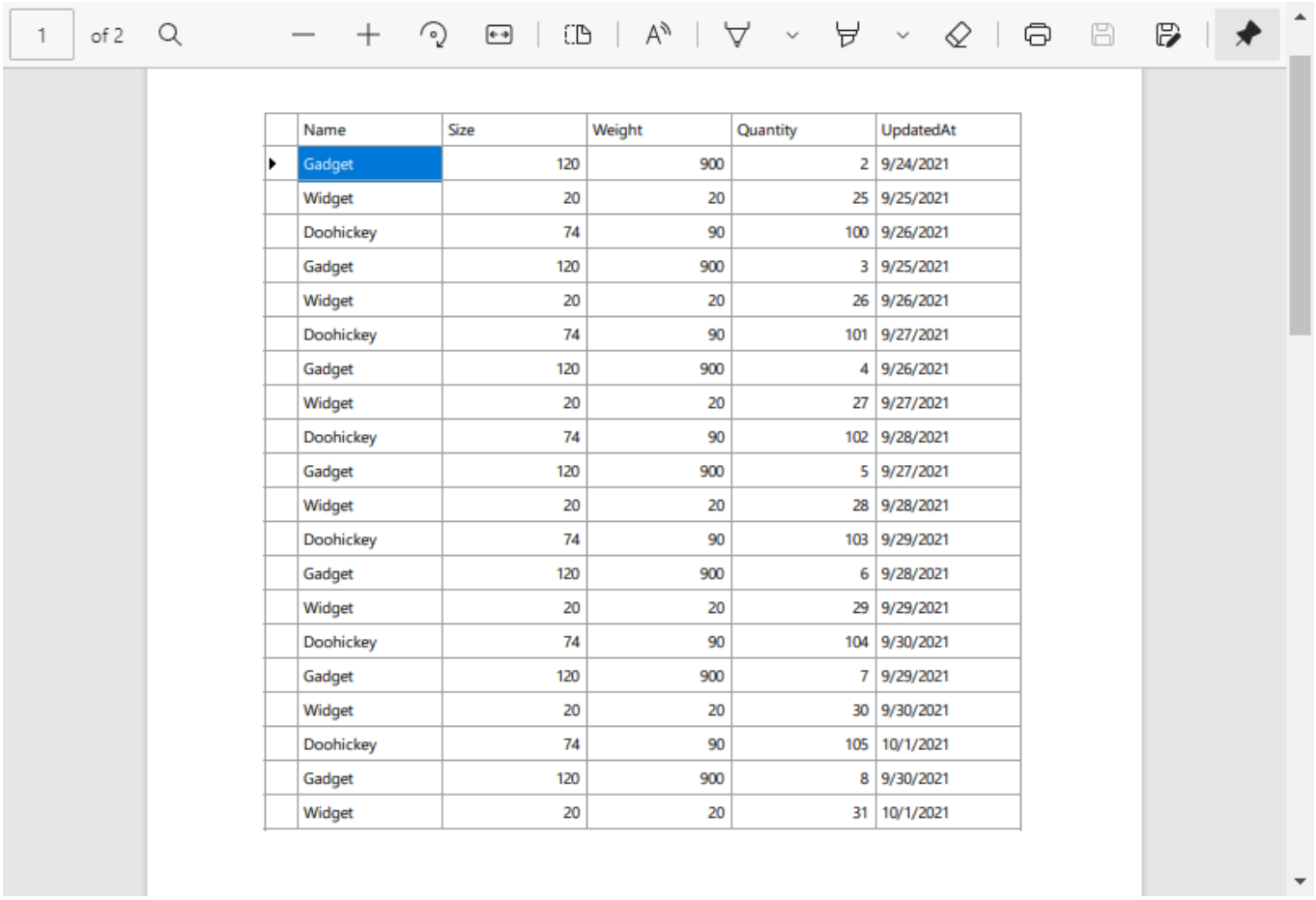
VB.NET

```
Dim printDocument = _flexGrid.PrintParameters.PrintDocument
printDocument.DocumentName = "Export to PDF"
printDocument.PrinterSettings.PrinterName = "Microsoft Print to PDF"
' ドキュメントをMicrosoftPDFプリンターに印刷します
printDocument.Print()
```

However, this approach has some limitations and it does not let you set advanced options such as setting borders, printing by specific page or column breaks etc. To implement these kind of advanced export options, you can use the C1PrintDocument class of the C1PrintDocument library shipped with ComponentOne WinForms Edition as shown in the section below.

Advanced Export to PDF

To export a grid to PDF with advanced options, you need to create images of grid by defining the rows and columns per page and hence, finding the row and column range to be drawn on each page. Then, add these images to generate a PDF using the C1PrintDocument class. Below are the detailed steps and sample code to implement the advanced export to PDF.



	Name	Size	Weight	Quantity	UpdatedAt
▸	Gadget	120	900	2	9/24/2021
	Widget	20	20	25	9/25/2021
	Doohickey	74	90	100	9/26/2021
	Gadget	120	900	3	9/25/2021
	Widget	20	20	26	9/26/2021
	Doohickey	74	90	101	9/27/2021
	Gadget	120	900	4	9/26/2021
	Widget	20	20	27	9/27/2021
	Doohickey	74	90	102	9/28/2021
	Gadget	120	900	5	9/27/2021
	Widget	20	20	28	9/28/2021
	Doohickey	74	90	103	9/29/2021
	Gadget	120	900	6	9/28/2021
	Widget	20	20	29	9/29/2021
	Doohickey	74	90	104	9/30/2021
	Gadget	120	900	7	9/29/2021
	Widget	20	20	30	9/30/2021
	Doohickey	74	90	105	10/1/2021
	Gadget	120	900	8	9/30/2021
	Widget	20	20	31	10/1/2021

Step 1: Create Ranges

First, you need to set the UserData property according to the defined rows per page and columns per page. This helps in finding out the row and column range to be drawn on each page of the PDF document. So, this step acts as a ground to create images of the grid in the step below.

C#

```
// get rows/cols per page
int rpp = 10;
int cpp = 3;
try
{
    rpp = int.Parse(_rpp.Text);
    cpp = int.Parse(_cpp.Text);
}
catch { }

// mark grid with row/column breaks
for (int r = _flex.Rows.Fixed; r < _flex.Rows.Count; r++)
{
    _flex.Rows[r].UserData = (r % rpp == 0)
        ? "*"
        : null;
}
for (int c = _flex.Cols.Fixed; c < _flex.Cols.Count; c++)
{
    _flex.Cols[c].UserData = (c % cpp == 0)
        ? "*"
        : null;
```

```
}
```

VB.NET

```
Dim rpp As Integer = 10
Dim cpp As Integer = 3
Try
    rpp = Integer.Parse(_rpp.Text)
    cpp = Integer.Parse(_cpp.Text)
Catch ex As Exception
End Try
For r As Integer = _flex.Rows.Fixed To _flex.Rows.Count - 1
    _flex.Rows(r).UserData = If((r Mod rpp = 0), "*", Nothing)
Next

For c As Integer = _flex.Cols.Fixed To _flex.Cols.Count - 1
    _flex.Cols(c).UserData = If((c Mod cpp = 0), "*", Nothing)
Next
```

Step 2: Create Grid Images

Now, create a custom class (in this case, FlexPDFCreator) to create images of the grid using CreateImage method of the **C1FlexGrid** class. The images are created based on the UserData property set in the step above.

C#

```
public class FlexPdfCreator
{
    // ** fields
    private ArrayList _images;
    private ArrayList _rowOnPage;
    private ArrayList _colOnPage;
    private int _rpp, _cpp;
    // ** ctor
    public FlexPdfCreator(C1FlexGrid flex, int _rpp, int _cpp)
    {
        // create page images
        _images = new ArrayList();
        _colOnPage = new ArrayList();
        _rowOnPage = new ArrayList();
        this._rpp = _rpp;
        this._cpp = _cpp;

        // initialize
        int r1, c1, r2, c2;

        // loop through columns looking for breaks
        c1 = c2 = flex.Cols.Fixed;
        for (int c = flex.Cols.Fixed; c < flex.Cols.Count; c++)
        {
            // check if this is a column break
            if (c == flex.Cols.Count - 1 ||
                (flex.Cols[c].UserData != null && flex.Cols[c].UserData.ToString()
== "*"))
            {
                // found break, column range is c1-c
                c2 = c;

                // loop through rows looking for breaks
                r1 = r2 = flex.Rows.Fixed;
                for (int r = flex.Rows.Fixed; r < flex.Rows.Count; r++)
```

```
                {
                    // look for next row break
                    if (r == flex.Rows.Count - 1 ||
                        (flex.Rows[r].UserData != null &&
flex.Rows[r].UserData.ToString() == "*"))
                    {
                        // found break, row range is r1-r
                        r2 = r;

                        // create image
                        _images.Add(flex.CreateImage(r1, c1, r2, c2));
                        _rowOnPage.Add(r2 - r1);
                        _colOnPage.Add(c2 - c1);

                        // update row range
                        r1 = r + 1;
                    }
                }

                // update column range
                c1 = c + 1;
            }
        }
    }
```

VB.NET

```
Public Class FlexPdfCreator
    Private _images As ArrayList
    Private _rowOnPage As ArrayList
    Private _colOnPage As ArrayList
    Private _rpp, _cpp As Integer
    Public Sub New(ByVal flex As C1FlexGrid, ByVal _rpp As Integer, ByVal _cpp As
Integer)
        _images = New ArrayList()
        _colOnPage = New ArrayList()
        _rowOnPage = New ArrayList()
        Me._rpp = _rpp
        Me._cpp = _cpp
        Dim r1, c1, r2, c2 As Integer
        c1 = flex.Cols.Fixed
        c2 = flex.Cols.Fixed
        For c As Integer = flex.Cols.Fixed To flex.Cols.Count - 1
            If c = flex.Cols.Count - 1 OrElse (flex.Cols(c).UserData IsNot Nothing
AndAlso flex.Cols(c).UserData.ToString() = "*") Then
                c2 = c
                r1 = flex.Rows.Fixed
                r2 = flex.Rows.Fixed
                For r As Integer = flex.Rows.Fixed To flex.Rows.Count - 1
                    If r = flex.Rows.Count - 1 OrElse (flex.Rows(r).UserData IsNot
Nothing AndAlso flex.Rows(r).UserData.ToString() = "*") Then
                        r2 = r
                        _images.Add(flex.CreateImage(r1, c1, r2, c2))
                        _rowOnPage.Add(r2 - r1)
                        _colOnPage.Add(c2 - c1)
                        r1 = r + 1
                    End If
                Next
                c1 = c + 1
            End If
        Next
    End Sub
```

Step 3: Generate PDF Document

In this step, create a document using C1PrintDocument class of the C1PrintDocument library shipped with ComponentOne WinForms Edition. Use the **RenderImage** class to add created images to this document and finally, save the PDF document using **Export** method.

C#

```
public void Save(string fileName)
{
    int perUnitWidth = 250 / _cpp;
    int perUnitHeight = 190 / _rpp;
    // create new C1PrintDocument
    C1PrintDocument c1PrintDocument1 = new C1PrintDocument();
    c1PrintDocument1.AllowNonReflowableDocs = true;
    RenderArea ra1 = new RenderArea();
    // add images to document
    for (int page = 0; page < _images.Count; page++)
    {
        Image img = _images[page] as Image;
        RenderImage ri1 = new RenderImage(img);
        ri1.Width = perUnitWidth * (int)_colOnPage[page] + "mm";
        ri1.Height = perUnitHeight * (int)_rowOnPage[page] + "mm";
        ra1.Children.Add(ri1);

    }
    c1PrintDocument1.Body.Children.Add(ra1);
    c1PrintDocument1.Reflow();
    c1PrintDocument1.Export(fileName);
}
```

VB.NET

```
Public Sub Save(ByVal fileName As String)
        Dim perUnitWidth As Integer = 250 / _cpp
        Dim perUnitHeight As Integer = 190 / _rpp
        Dim c1PrintDocument1 As C1PrintDocument = New C1PrintDocument()
        c1PrintDocument1.AllowNonReflowableDocs = True
        Dim ra1 As RenderArea = New RenderArea()

        For page As Integer = 0 To _images.Count - 1
            Dim img As Image = TryCast(_images(page), Image)
            Dim ri1 As RenderImage = New RenderImage(img)
            ri1.Width = perUnitWidth * CInt(_colOnPage(page)) & "mm"
            ri1.Height = perUnitHeight * CInt(_rowOnPage(page)) & "mm"
            ra1.Children.Add(ri1)
        Next
        c1PrintDocument1.Body.Children.Add(ra1)
        c1PrintDocument1.Reflow()
        c1PrintDocument1.Export(fileName)
    End Sub
End Class
```

ロード

FlexGrid では、グリッドをさまざまな形式に保存できるだけでなく、テキスト、Excel、XML、データベースなどのさまざまな形式からデータをロードできます。この機能は、**C1.Win.C1FlexGrid.ImportExport.dll** という名前のアセンブリを通じて有効になります。したがって、これらのソースからグリッドにデータをロードするには、このアセンブリへの参照を追加する必要があります。

テキストファイルからのロード

テキストファイルからデータをロードするために、FlexGrid には **Extensions** クラスの **LoadGrid** メソッドが用意されています。このメソッドには、区切り文字やエンコードなどを選択するためのパラメータがあります。**SaveGrid** メソッドを使用して保存されたテキストファイルをロードすることもできます。テキストファイルをロードすると、ファイルの内容に合わせて、必要に応じて行と列がグリッドに追加されます。このメソッドは、カンマ区切りテキストファイル (CSV 形式)、タブ区切りテキストファイルなどの形式のほか、MS Excel ファイル(.xls)もサポートしています。

次のコードは、テキストファイルからデータをロードして WinForms FlexGrid に挿入する方法を示します。

C#

```
private void btnLoadTxt_Click(object sender, EventArgs e)
{
    c1FlexGrid1.LoadGrid("../.../ExportedGrid.txt", FileFormatEnum.TextComma );
}
```

VB.NET

```
Private Sub btnLoadTxt_Click(ByVal sender As Object, ByVal e As EventArgs)
    c1FlexGrid1.LoadGrid("../.../ExportedGrid.txt", FileFormatEnum.TextComma)
End Sub
```

Excel ファイルからのロード

Excel ファイルからグリッドをロードするには、前述の **LoadGrid** メソッドを使用し、FileFormatEnum.Excel に書式パラメータを設定します。コンピュータに Microsoft Excel をインストールしておく必要はありません。ただし、LoadGrid メソッドは、ワークブックの最初のワークシートからしかデータをロードできません。

Excel ファイルからデータをロードする際に、さらに詳細な制御を行うには、代わりに **LoadExcel** メソッドを使用します。このメソッドで Excel ファイルをロードすると、行や列のサイズ、フォント、色、書式、セルの配置など、ほとんどのデータ型と書式設定情報が変換されます。ただし、いくつかの例外はあります。たとえば、グリッドは Excel のセル内の値をロードしますが、その基になる式をロードすることはできません。ほかにも、固定セル、結合セル、画像、データマップ、セル境界線などの機能も変換されません。

Excel ファイルから WinForms FlexGrid にコンテンツをロードするには、次のコードを使用します。

C#

```
private void btnLoadExcl_Click(object sender, EventArgs e)
{
    c1FlexGrid1.LoadExcel("../.../ExportedGrid.xlsx");
}
```

VB.NET

```
Private Sub btnLoadExcl_Click(ByVal sender As Object, ByVal e As EventArgs)
    c1FlexGrid1.LoadExcel("../.../ExportedGrid.xlsx")
End Sub
```

データベースからのロード

データベースからグリッドデータをロードするには、**DataReader** オブジェクトを使用します。このプロセスはデータ連結とは異なります。つまり、1 つ以上のコントロールと基底のデータソースの間に有効な接続は維持されません。

次のコードは、データベースから WinForms FlexGrid にコンテンツをロードする方法を示しています。

C#

```
private void btnLoadDB_Click(object sender, EventArgs e)
{

    {
        // DataReaderを準備します。
        string strConn = "data source=MYMACHINE;initial catalog=Northwind;";
        System.Data.SqlClient.SqlConnection myConn = new
System.Data.SqlClient.SqlConnection(strConn);
        System.Data.SqlClient.SqlCommand myCMD = new
System.Data.SqlClient.SqlCommand("SELECT * FROM Employees", myConn);
        myConn.Open();
        System.Data.SqlClient.SqlDataReader myReader = myCMD.ExecuteReader();

        // DBスキーマからグリッド構造を構築します。
        DataTable dt = myReader.GetSchemaTable();
       c1FlexGrid1.Cols.Count = 1;
        foreach (DataRow dr in dt.Rows)
        {
            Column c =c1FlexGrid1.Cols.Add();
            c.Caption = c.Name = (string)dr["ColumnName"];
            c.DataType = (Type)dr["DataType"];
        }

        // グリッドにデータを入力します。
       c1FlexGrid1.Rows.Count = 1;
        int row = 1;
        int cols = dt.Columns.Count;
        object[] v = (object[])Array.CreateInstance(typeof(object), cols);
        while (myReader.Read())
        {
            myReader.GetValues(v);
           c1FlexGrid1.AddItem(v, row++, 1);
        }

        // クリーンアップします。
       c1FlexGrid1.AutoSizeCols();
        myReader.Close();
        myConn.Close();
    }
}
```

VB.NET

```
Private Sub btnLoadDB_Click(ByVal sender As Object, ByVal e As EventArgs)

    '  DataReaderを準備します。
    Dim strConn As String = "data source=MYMACHINE;initial catalog=Northwind;"
    Dim myConn As New SqlClient.SqlConnection(strConn)
    Dim myCMD As New SqlClient.SqlCommand("SELECT * FROM Employees", myConn)
    myConn.Open()
    Dim myReader As SqlClient.SqlDataReader = myCMD.ExecuteReader()

    ' DBスキーマからグリッド構造を構築します。
    Dim dt As DataTable = myReader.GetSchemaTable()
    _flex.Cols.Count = 1
    Dim dr As DataRow
    For Each dr In  dt.Rows
        Dim c As C1.Win.C1FlexGrid.Column = _flex.Cols.Add()
        c.Caption =(c.Name <= CStr(dr("ColumnName")))
        c.DataType = CType(dr("DataType"), Type)
```

```
    Next dr

    '  グリッドにデータを入力します。
    _flex.Rows.Count = 1
    Dim row As Integer = 1
    Dim cols As Integer = dt.Columns.Count
    Dim v As Object() = CType(Array.CreateInstance(GetType(Object), cols), Object())
    While myReader.Read()
        myReader.GetValues(v)
        _flex.AddItem(v, row + 1, 1)
    End While

    '  クリーンアップします。
    _flex.AutoSizeCols()
    myReader.Close()
    myConn.Close()
End Sub
```

XML からロード

グリッドのコンテンツを XML ドキュメントからシリアライズ解除するには、**C1FlexGrid** クラスの **ReadXML** メソッドを呼び出し、XML ドキュメントのパスをパラメータとして渡します。

XML ドキュメントから WinForms FlexGrid にコンテンツをロードするには、次のコードを使用します。

C#

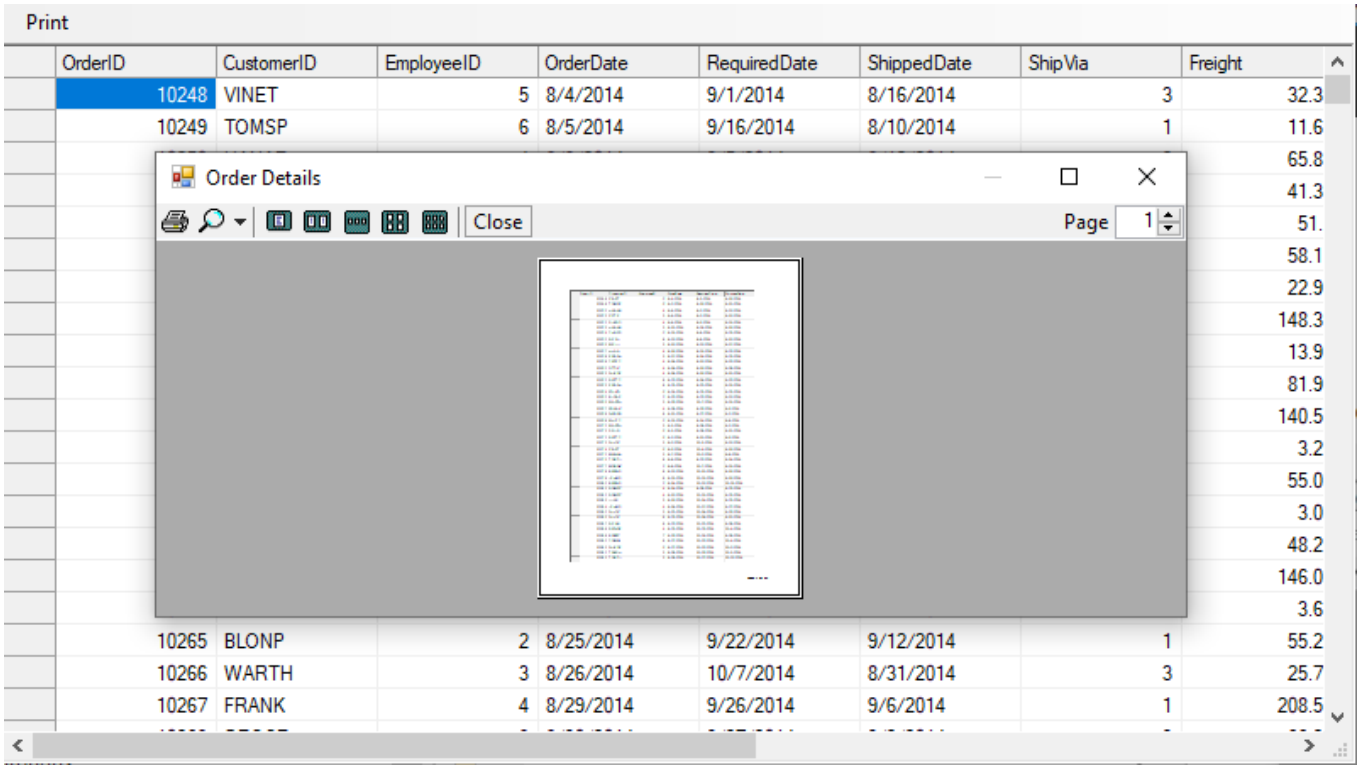
```
private void btnLoadXML_Click(object sender, EventArgs e)
{
    c1FlexGrid1.ReadXml("../.../ExportedGrid.xml");
}
```

VB.NET

```
Private Sub btnLoadXML_Click(ByVal sender As Object, ByVal e As EventArgs)
    c1FlexGrid1.ReadXml("../.../ExportedGrid.xml")
End Sub
```

印刷

FlexGrid では、グリッドを印刷したり、組み込みのメソッドやプロパティを使用して印刷の基本設定や詳細設定を行うことができます。



グリッドの印刷

FlexGrid では、**C1FlexGrid** クラスの **PrintGrid** メソッドを使用して、グリッドの内容を基本的な印刷オプションで印刷することができます。このメソッドには、オプションの **PrintGridFlags** パラメータがあり、スケーリングモードなどのグリッドの印刷方法や、印刷関連のさまざまなダイアログボックスを表示するかどうかを指定できます。このメソッドを使用して、印刷するグリッドのヘッダーやフッターにテキストを設定することもできます。

次のコードは、PrintGrid メソッドを使用して WinForms FlexGrid を印刷する方法を示します。

C#

```
// プレビューダイアログを表示し、指定したヘッダとフッターでグリッドを印刷します
c1FlexGrid1.PrintGrid("C1FlexGrid", PrintGridFlags.FitToPageWidth |
PrintGridFlags.ShowPreviewDialog, "C1FlexGrid\t\t" +
String.Format(DateTime.Now.ToString(), "d"), "\t\tPage {0} of {1}");
```

VB.NET

```
' プレビューダイアログを表示し、指定したヘッダとフッターでグリッドを印刷します
c1FlexGrid1.PrintGrid("C1FlexGrid", PrintGridFlags.FitToPageWidth Or
PrintGridFlags.ShowPreviewDialog, "C1FlexGrid" & vbTab & vbTab &
String.Format(Date.Now.ToString(), "d"), vbTab & vbTab & "Page {0} of {1}")
```

印刷オプション

ヘッダーやフッターのフォント、ページの余白、ページの向きなどの詳細な印刷オプションを設定するには、**C1FlexGrid** クラスの **PrintParameter** プロパティを使用します。

詳細な印刷オプションで WinForms FlexGrid を印刷するには、次のコードを使用します。

C#

```
// グリッドのPrintDocumentオブジェクトを取得します .
```

```
System.Drawing.Printing.PrintDocument pd =
c1FlexGrid1.PrintParameters.PrintDocument;
// ページを設定します(横向き、1.5 "左マージン)。
pd.DefaultPageSettings.Landscape = true;
pd.DefaultPageSettings.Margins.Left = 150;
// ヘッダとフッタのフォントを設定します
c1FlexGrid1.PrintParameters.HeaderFont = new Font("Arial Black", 14,
FontStyle.Bold);
c1FlexGrid1.PrintParameters.FooterFont = new Font("Arial Narrow", 8,
FontStyle.Italic);
```

VB.NET

```
' グリッドのPrintDocumentオブジェクトを取得します
Dim pd As Drawing.Printing.PrintDocument = c1FlexGrid1.PrintParameters.PrintDocument
' ページを設定します(横向き、1.5 "左マージン)。
pd.DefaultPageSettings.Landscape = True
pd.DefaultPageSettings.Margins.Left = 150
' ヘッダとフッタのフォントを設定します
c1FlexGrid1.PrintParameters.HeaderFont = New Font("Arial Black", 14, FontStyle.Bold)
c1FlexGrid1.PrintParameters.FooterFont = New Font("Arial Narrow", 8,
FontStyle.Italic)
```

印刷プレビューダイアログのカスタマイズ

GridPrinter クラスの **PrintPreviewDialog** プロパティを使用して、印刷プレビューダイアログをカスタマイズできます。このプロパティには、**C1FlexGrid** クラスの **PrintParameter** プロパティからアクセスできます。次のコードは、PrintPreviewDialog プロパティを使用して、独自のキャプションを付けたプレビューダイアログを最大化して表示します。

次のコードは、WinForms FlexGrid の印刷プレビューダイアログをカスタマイズする方法を示しています。

C#

```
Form dlg = c1FlexGrid1.PrintParameters.PrintPreviewDialog as Form;
dlg.Text = "Order Details";
dlg.StartPosition = FormStartPosition.CenterParent;
dlg.WindowState = FormWindowState.Maximized;
c1FlexGrid1.PrintGrid("Orders", PrintGridFlags.ShowPreviewDialog);
```

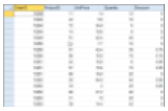
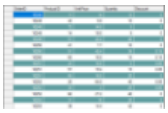
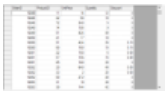
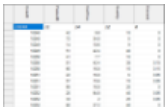
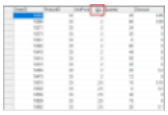
VB.NET

```
Dim dlg As Form = TryCast(c1FlexGrid1.PrintParameters.PrintPreviewDialog, Form)
dlg.Text = "Order Details"
dlg.StartPosition = FormStartPosition.CenterParent
dlg.WindowState = FormWindowState.Maximized
c1FlexGrid1.PrintGrid("Orders", PrintGridFlags.ShowPreviewDialog)
```

スタイル設定と外観

FlexGrid は、アプリケーションのニーズと外観に合わせてグリッドとグリッドの要素をスタイル設定できるように、組み込みのビジュアルスタイルのほか、さまざまな設計時オプションと実行時オプションを提供しています。また、コードを 1 行も記述することなくグリッドの外観をカスタマイズできるように、スタイルエディタも提供されています。これらのエディタおよび設計時のスタイル設定の詳細については、「エディタ」を参照してください。

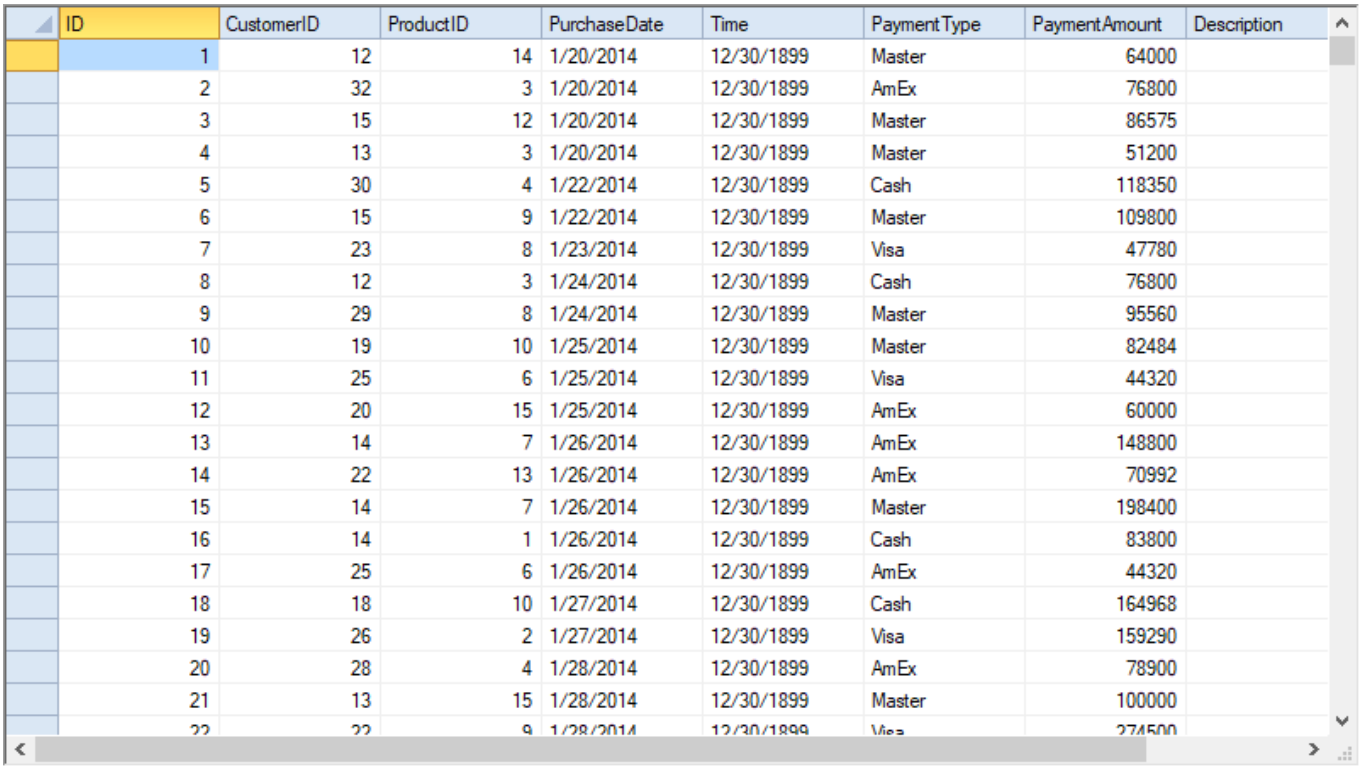
このセクションでは、FlexGrid によって提供される組み込みオプション、およびグリッドの外観とその要素をカスタマイズするさまざまな方法について説明します。

トピック	スナップショット	コンテンツ
組み込みオプション		用意されているビジュアルスタイルおよびスタイルの組み込みコレクションについて説明します。 • ビジュアルスタイル • スタイルのコレクション
カスタムスタイル		カスタムスタイルを作成して適用する方法について説明します。 • 方法 1:オブジェクトのスタイル設定 • 方法 2:再利用可能なスタイルの作成
グリッドのカスタマイズ		グリッドレベルのカスタマイズについて説明します。 • 交互表示行の設定 • 背景画像の設定
境界線のカスタマイズ		行、列、またはグリッドレベルでのセル境界線のカスタマイズについて説明します。 • グリッドの境界線のカスタマイズ • 行/列の境界線のカスタマイズ • セルの境界線のカスタマイズ
テキストのカスタマイズ		グリッドテキストのスタイル設定のさまざまな側面について説明します。 • フォントの変更 • マージンの設定 • 縦書きテキストの設定 • テキストの折り返し • トリミングされたテキストの表示 • テキストの配置 • テキストエフェクトの適用
カスタムグリフ		グリッド内のカスタムグリフの適用について説明します。

組み込みオプション

ビジュアルスタイル

Microsoft Office 2007 および 2010 のテーマに基づいて FlexGrid の外観を簡単にカスタマイズできるように、FlexGrid は 6 つの組み込みビジュアルスタイルを備えています。これらのビジュアルスタイルには、C1FlexGrid クラスの **VisualStyle** プロパティを通してアクセスできます。MS Office ベースのビジュアルスタイルとは別に、このプロパティには Custom と System を設定できます。**Custom** を設定すると、グリッドはビジュアルスタイルを適用せず、グリッドのプロパティを使用してレンダリングします。このプロパティに **System** を設定すると、グリッドは現在のシステム設定に基づいて外観をレンダリングします。



ID	CustomerID	ProductID	PurchaseDate	Time	PaymentType	PaymentAmount	Description
1	12	14	1/20/2014	12/30/1899	Master	64000	
2	32	3	1/20/2014	12/30/1899	AmEx	76800	
3	15	12	1/20/2014	12/30/1899	Master	86575	
4	13	3	1/20/2014	12/30/1899	Master	51200	
5	30	4	1/22/2014	12/30/1899	Cash	118350	
6	15	9	1/22/2014	12/30/1899	Master	109800	
7	23	8	1/23/2014	12/30/1899	Visa	47780	
8	12	3	1/24/2014	12/30/1899	Cash	76800	
9	29	8	1/24/2014	12/30/1899	Master	95560	
10	19	10	1/25/2014	12/30/1899	Master	82484	
11	25	6	1/25/2014	12/30/1899	Visa	44320	
12	20	15	1/25/2014	12/30/1899	AmEx	60000	
13	14	7	1/26/2014	12/30/1899	AmEx	148800	
14	22	13	1/26/2014	12/30/1899	AmEx	70992	
15	14	7	1/26/2014	12/30/1899	Master	198400	
16	14	1	1/26/2014	12/30/1899	Cash	83800	
17	25	6	1/26/2014	12/30/1899	AmEx	44320	
18	18	10	1/27/2014	12/30/1899	Cash	164968	
19	26	2	1/27/2014	12/30/1899	Visa	159290	
20	28	4	1/28/2014	12/30/1899	AmEx	78900	
21	13	15	1/28/2014	12/30/1899	Master	100000	
22	22	9	1/28/2014	12/30/1899	Visa	274500	

このプロパティは、デザイナを使用して、またはコードを通して設定できます。設計時にプロパティを設定するには、スマートタグをクリックして **C1FlexGrid タスク**メニューを開き、**[VisualStyle]**コンボボックスからビジュアルスタイルを選択します。コードを通して WinForms FlexGrid にビジュアルスタイルを適用するには、以下のコードを使用します。

C#

```
// 視覚スタイルをOffice2010Blueスキームに設定します
c1FlexGrid1.VisualStyle = VisualStyle.Office2010Blue;
```

VB.NET

```
' 視覚スタイルをOffice2010Blueスキームに設定します
c1FlexGrid1.VisualStyle = VisualStyle.Office2010Blue
```

スタイルのコレクション

FlexGrid は、通常セル、固定セル、フォーカスセルなど、定義済みのセルタイプや状態に使用されるスタイルの組み込みコレクションを提供しています。設計時およびコードからこれらの組み込みスタイルを使用できます。デザインビューで、C1FlexGrid スタイルエディタ からこれらのスタイルにアクセスできます。このエディタは、タスクメニューで **[スタイル]** オプションをクリックして開くことができます。このスタイルのコレクションは **CellStyleCollection** クラスによって表され、**C1FlexGrid** クラスの **Styles** プロパティを通してアクセスできます。各組み込みスタイルは CellStyle クラスのオブジェクトであり、**BackColor**、**DataType**、**Format** などのまざまなプロパティを持っています。

以下のコードは、WinForms FlexGrid の Styles コレクションからスタイルを使用する方法を示しています。

C#

```
// 組み込みセルスタイルNormalのプロパティを設定します
c1FlexGrid1.Styles["Normal"].BackColor = Color.Azure;
c1FlexGrid1.Styles["Normal"].ForeColor = Color.BlueViolet;
// 固定セルの背景色を設定します
c1FlexGrid1.Styles["Fixed"].BackColor = Color.Aqua;
```

VB.NET

```
' 組み込みセルスタイルNormalのプロパティを設定します
c1FlexGrid1.Styles("Normal").BackColor = Color.Azure
c1FlexGrid1.Styles("Normal").ForeColor = Color.BlueViolet
' 固定セルの背景色を設定します
c1FlexGrid1.Styles("Fixed").BackColor = Color.Aqua
```

Add メソッドを使用して、独自のカスタムスタイルを Styles コレクションに追加することもできます。カスタムスタイルを作成してグリッドセルに適用する方法の詳細については、「カスタムスタイル」を参照してください。

カスタムスタイル

方法 1:オブジェクトのスタイル設定

WinForms FlexGrid の特定の行、列、またはセル範囲のスタイルを設定するには、行、列、またはセル範囲オブジェクトの StyleNew プロパティを使用できます。

C#

```
// 特定の行にカスタムスタイルを適用します
c1FlexGrid1.Rows[1].StyleNew.BackColor = Color.Azure;
c1FlexGrid1.Rows[1].StyleNew.ForeColor = Color.BlueViolet;
```

VB.NET

```
' 特定の行にカスタムスタイルを適用します
c1FlexGrid1.Rows(1).StyleNew.BackColor = Color.Azure
c1FlexGrid1.Rows(1).StyleNew.ForeColor = Color.BlueViolet
```

このアプローチは、スタイルを再利用する必要がない場合に、特定のオブジェクトにスタイルを設定するために便利です。特定のスタイルを再利用するには、次のセクションで説明するように、CellStyle オブジェクトを使用してスタイルを作成する必要があります。

方法 2:再利用可能なスタイルの作成

このアプローチでは、カスタムスタイルを **CellStyle** クラスのオブジェクトとして作成し、**Add** メソッドを使用してそれを **Styles** コレクションに追加します。次に、そのプロパティを定義し、必要に応じて行、列、またはセル範囲に適用します。上記のように、このアプローチは、特定のスタイルを繰り返し使用する必要がある場合にたいへん便利です。

以下のコードを使用して、WinForms FlexGrid 用に再利用可能なカスタムスタイルを作成します。

C#

```
// 新しいスタイルを作成し、スタイルコレクションに追加します
CellStyle cs = this.c1FlexGrid1.Styles.Add("Custom");

// 新しいカスタムスタイルのプロパティを定義します
cs.BackColor = Color.Azure;
cs.ForeColor = Color.BlueViolet;
// 行にカスタムスタイルを適用します
c1FlexGrid1.Rows[1].Style = cs;
```

VB.NET

```
' 新しいスタイルを作成し、スタイルコレクションに追加します
Dim cs As CellStyle = Me.c1FlexGrid1.Styles.Add("Custom")
```

```
' 新しいカスタムスタイルのプロパティを定義します
cs.BackColor = Color.Azure
cs.ForeColor = Color.BlueViolet
' 行にカスタムスタイルを適用します
c1FlexGrid1.Rows(1).Style = cs
```

グリッドのカスタマイズ

FlexGrid では、グリッドの外観を全体的にカスタマイズできますが、それによって見栄えをよくするだけでなく、可読性も向上させることができます。交互表示行を追加することで、グリッドの可読性が向上します。グリッドの背景に、ウォーターマークや企業ロゴなどの画像を設定することもできます。

交互表示行の設定

グリッドに交互表示行の色とスタイルを設定するには、設計時または実行時に組み込みスタイルの「Alternate」を使用できます。設計時にスタイルを適用するには、FlexGrid スタイルエディタにアクセスし、Alternate スタイルのプロパティを設定します。

	ID	CustomerID	ProductID	PurchaseDate	Time	PaymentType	PaymentAmount	Description
	1	12	14	1/20/2014	12/30/1899	Master	64000	
	2	32	3	1/20/2014	12/30/1899	AmEx	76800	
	3	15	12	1/20/2014	12/30/1899	Master	86575	
	4	13	3	1/20/2014	12/30/1899	Master	51200	
	5	30	4	1/22/2014	12/30/1899	Cash	118350	
	6	15	9	1/22/2014	12/30/1899	Master	109800	
	7	23	8	1/23/2014	12/30/1899	Visa	47780	
	8	12	3	1/24/2014	12/30/1899	Cash	76800	
	9	29	8	1/24/2014	12/30/1899	Master	95560	
	10	19	10	1/25/2014	12/30/1899	Master	82484	
	11	25	6	1/25/2014	12/30/1899	Visa	44320	
	12	20	15	1/25/2014	12/30/1899	AmEx	60000	
	13	14	7	1/26/2014	12/30/1899	AmEx	148800	
	14	22	13	1/26/2014	12/30/1899	AmEx	70992	
	15	14	7	1/26/2014	12/30/1899	Master	198400	
	16	14	1	1/26/2014	12/30/1899	Cash	83800	
	17	25	6	1/26/2014	12/30/1899	AmEx	44320	
	18	18	10	1/27/2014	12/30/1899	Cash	164968	
	19	26	2	1/27/2014	12/30/1899	Visa	159290	
	20	28	4	1/28/2014	12/30/1899	AmEx	78900	
	21	13	15	1/28/2014	12/30/1899	Master	100000	
	22	22	9	1/28/2014	12/30/1899	Visa	274500	

コードから WinForms FlexGrid に交互表示行スタイルを適用するには、CellStyleCollection クラスの CellStyle 項目「Alternate」を使用し、交互表示スタイルを設定するためのさまざまなプロパティを設定します。

C#

```
// 代替行の前色を設定します
 c1FlexGrid1.Styles["Alternate"].ForeColor = Color.White;
 // 代替行の背景色を設定します
 c1FlexGrid1.Styles["Alternate"].BackColor = Color.CadetBlue;
```

VB.NET

```
' 代替行の前色を設定します
c1FlexGrid1.Styles("Alternate").ForeColor = Color.White
' 代替行の背景色を設定します
c1FlexGrid1.Styles("Alternate").BackColor = Color.CadetBlue
```

グリッドの背景画像の設定

グリッドに背景画像を設定するには、BackgroundImage プロパティを使用して、画像ファイルのパスを割り当てます。**BackgroundImageLayout** という名前のもう 1 つのプロパティを使用して、グリッドに画像をレンダリングするかどうかとその方法を選択できます。

以下のコードを使用して、WinForms FlexGrid に背景画像を設定します。

C#

```
// 背景画像をグリッドに設定します
c1FlexGrid1.BackgroundImage = Image.FromFile("C:\\IMG-20190524-WA0037.png");
c1FlexGrid1.BackgroundImageLayout = ImageLayout.Stretch;
```

VB.NET

```
' 背景画像をグリッドに設定します
c1FlexGrid1.BackgroundImage = Image.FromFile("C:\IMG-20190524-WA0037.png")
c1FlexGrid1.BackgroundImageLayout = ImageLayout.Stretch
```

境界線のカスタマイズ

FlexGrid コントロールでは、境界線のスタイル、色、方向などを変更して、グリッド、行、列、さらにはセルの境界線をカスタマイズできます。

ID	CustomerID	ProductID	PurchaseDate	Time	PaymentType	PaymentAmount	Description
1	12	14	1/20/2014	12/30/1899	Master	64000	
2	32	3	1/20/2014	12/30/1899	AmEx	76800	
3	15	12	1/20/2014	12/30/1899	Master	86575	
4	13	3	1/20/2014	12/30/1899	Master	51200	
5	30	4	1/22/2014	12/30/1899	Cash	118350	
6	15	9	1/22/2014	12/30/1899	Master	109800	
7	23	8	1/23/2014	12/30/1899	Visa	47780	
8	12	3	1/24/2014	12/30/1899	Cash	76800	
9	29	8	1/24/2014	12/30/1899	Master	95560	
10	19	10	1/25/2014	12/30/1899	Master	82484	
11	25	6	1/25/2014	12/30/1899	Visa	44320	
12	20	15	1/25/2014	12/30/1899	AmEx	60000	
13	14	7	1/26/2014	12/30/1899	AmEx	148800	
14	22	13	1/26/2014	12/30/1899	AmEx	70992	
15	14	7	1/26/2014	12/30/1899	Master	198400	
16	14	1	1/26/2014	12/30/1899	Cash	83800	
17	25	6	1/26/2014	12/30/1899	AmEx	44320	
18	18	10	1/27/2014	12/30/1899	Cash	164968	
19	26	2	1/27/2014	12/30/1899	Visa	159290	
20	28	4	1/28/2014	12/30/1899	AmEx	78900	
21	13	15	1/28/2014	12/30/1899	Master	100000	
22	22	9	1/28/2014	12/30/1899	Visa	274500	

グリッドの境界線のカスタマイズ

グリッドコントロールの境界線をカスタマイズするには、**BorderStyle** プロパティを使用します。このプロパティは、C1.Win.FlexGrid.Util.BaseControls 名前空間にある **BorderStyleEnum** の値を受け取ります。

以下のコードは、WinForms FlexGrid コントロールの境界線をカスタマイズする方法を示します。

C#

```
// グリッドの境界線を3次元の境界線に変更します
c1FlexGrid1.BorderStyle =
C1.Win.C1FlexGrid.Util.BaseControls.BorderStyleEnum.Fixed3D;
```

VB.NET

```
' グリッドの境界線を3次元の境界線に変更します
c1FlexGrid1.BorderStyle =
C1.Win.C1FlexGrid.Util.BaseControls.BorderStyleEnum.Fixed3D
```

行/列の境界線のカスタマイズ

特定の行または列の境界線をカスタマイズするには、StyleNew プロパティを使用して CellStyle クラスの **'Border'** 項目にアクセスし、境界線のスタイル、方向、色などのプロパティを設定する必要があります。グリッドコントロールには **BorderStyleEnum** と **BorderDirEnum** があり、それぞれ境界線のスタイルと方向を設定します。

以下のコードを使用して、WinForms FlexGrid の行または列の境界線を変更します。

C#

```
// 最初の列の境界線スタイルを設定します
c1FlexGrid1.Cols[1].StyleNew.Border.Style = BorderStyleEnum.Groove;
c1FlexGrid1.Cols[1].StyleNew.Border.Color = Color.Red;
c1FlexGrid1.Cols[1].StyleNew.Border.Direction = BorderDirEnum.Vertical;
//最初の行の境界線スタイルを設定します
c1FlexGrid1.Rows[1].StyleNew.Border.Style = BorderStyleEnum.Raised;
c1FlexGrid1.Rows[1].StyleNew.Border.Color = Color.Blue;
```

VB.NET

```
' 最初の列の境界線スタイルを設定します
c1FlexGrid1.Cols(1).StyleNew.Border.Style = BorderStyleEnum.Groove
c1FlexGrid1.Cols(1).StyleNew.Border.Color = Color.Red
c1FlexGrid1.Cols(1).StyleNew.Border.Direction = BorderDirEnum.Vertical
'最初の行の境界線スタイルを設定します
c1FlexGrid1.Rows(1).StyleNew.Border.Style = BorderStyleEnum.Raised
c1FlexGrid1.Rows(1).StyleNew.Border.Color = Color.Blue
```

セルの境界線のカスタマイズ

グリッド内のすべてのセルの境界線をカスタマイズするには、組み込みスタイル "**Normal**" にアクセスし、その境界線プロパティを設定します。同様に、固定セル、フリーズセルなど、特定のタイプのセルのスタイルを変更するには、Styles コレクション内の対応するスタイルにアクセスします。

以下のコードは、WinForms FlexGrid の通常のセルの境界線をカスタマイズします。

C#

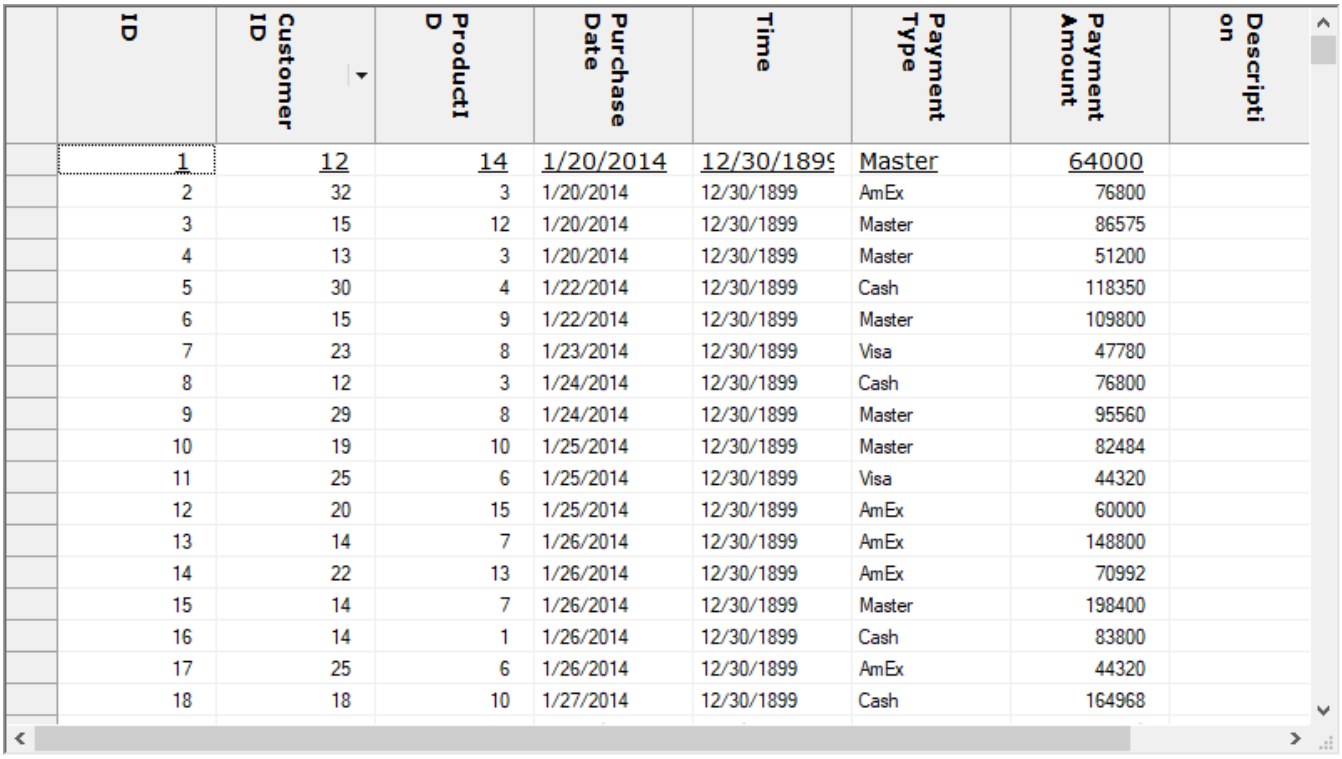
```
// すべてのグリッドセルの境界線スタイルを設定します
c1FlexGrid1.Styles.Normal.Border.Style = BorderStyleEnum.Double;
```

VB.NET

```
' すべてのグリッドセルの境界線スタイルを設定します
c1FlexGrid1.Styles.Normal.Border.Style = BorderStyleEnum.Double
```

テキストのカスタマイズ

FlexGrid では、フォント、マージン、方向、配置、エフェクトの変更など、さまざまな方法でテキストをカスタマイズできます。このトピックでは、特定の行オブジェクトを使用して、テキストスタイルの各部を設定する方法を示します。一方、既存の組み込みスタイルを使用する場合、または独自のカスタムスタイルを作成して再利用する場合は、対応するトピックを参照してください。



ID	Customer ID	Product ID	Purchase Date	Time	Payment Type	Payment Amount	Description
1	12	14	1/20/2014	12/30/1899	Master	64000	
2	32	3	1/20/2014	12/30/1899	AmEx	76800	
3	15	12	1/20/2014	12/30/1899	Master	86575	
4	13	3	1/20/2014	12/30/1899	Master	51200	
5	30	4	1/22/2014	12/30/1899	Cash	118350	
6	15	9	1/22/2014	12/30/1899	Master	109800	
7	23	8	1/23/2014	12/30/1899	Visa	47780	
8	12	3	1/24/2014	12/30/1899	Cash	76800	
9	29	8	1/24/2014	12/30/1899	Master	95560	
10	19	10	1/25/2014	12/30/1899	Master	82484	
11	25	6	1/25/2014	12/30/1899	Visa	44320	
12	20	15	1/25/2014	12/30/1899	AmEx	60000	
13	14	7	1/26/2014	12/30/1899	AmEx	148800	
14	22	13	1/26/2014	12/30/1899	AmEx	70992	
15	14	7	1/26/2014	12/30/1899	Master	198400	
16	14	1	1/26/2014	12/30/1899	Cash	83800	
17	25	6	1/26/2014	12/30/1899	AmEx	44320	
18	18	10	1/27/2014	12/30/1899	Cash	164968	

フォントの変更

特定の行オブジェクトまたは列オブジェクト内のテキストのフォントを変更するには、その CellStyle の **Font** プロパティを使用できます。以下のコードは、WinForms FlexGrid の行のフォントを変更する方法を示しています。

C#

```
//カスタムスタイルを特定の行に適用します
c1FlexGrid1.Rows[1].StyleNew.Font = new Font("verdana", 10, FontStyle.Underline);
c1FlexGrid1.Rows[0].StyleNew.Font = new Font("verdana", 10, FontStyle.Bold);
```

VB.NET

```
'カスタムスタイルを特定の行に適用します
c1FlexGrid1.Rows(1).StyleNew.Font = new Font("verdana", 10, FontStyle.Underline)
c1FlexGrid1.Rows(0).StyleNew.Font = new Font("verdana", 10, FontStyle.Bold)
```

一方、グリッド全体のフォントを変更する場合は、C1FlexGrid クラスの Font プロパティを設定します。以下のコードは、WinForms FlexGrid 全体のフォントを変更する方法を示しています。

C#

```
// グリッドの内容全体のフォントを変更します
c1FlexGrid1.Font = new Font("verdana", 10, FontStyle.Italic);
```

VB.NET

```
' グリッドの内容全体のフォントを変更します
```

```
c1FlexGrid1.Font = New Font("verdana", 10, FontStyle.Italic)
```

マージンの設定

特定の行または列のテキストにマージンを設定するには、**CellStyle** の **Margins** プロパティを設定します。以下のコードを使用して、WinForms FlexGrid の行のマージンを変更します。

C#

```
// 右側のマージンを設定します
c1FlexGrid1.Rows[1].StyleNew.Margins.Right = 10;
```

VB.NET

```
' 右側のマージンを設定します
c1FlexGrid1.Rows(1).StyleNew.Margins.Right = 10
```

縦書きテキストの設定

テキストの方向を変更して縦書きテキストとして表示するには、オブジェクトの CellStyle の **TextDirection** プロパティを設定します。また、縦書きテキストを正しく表示するには、テキストの長さに応じて対象のセルの高さを調整する必要がある場合があります。以下のコードでは、WinForms FlexGrid の行に縦書きテキストを設定します。

C#

```
// 行の内容のテキスト方向を設定します
c1FlexGrid1.Rows[1].StyleNew.TextDirection = TextDirectionEnum.Down;

//行の高さを設定して、垂直方向のテキストを表示します
c1FlexGrid1.Rows[1].Height = 60;
```

VB.NET

```
' 行の内容のテキスト方向を設定します
c1FlexGrid1.Rows(1).StyleNew.TextDirection = TextDirectionEnum.Down
'行の高さを設定して、垂直方向のテキストを表示します
c1FlexGrid1.Rows(1).Height = 60
```

テキストの折り返し

使用可能なセル幅より長いテキストを折り返すには、その **CellStyle** の WordWrap プロパティに **true** を設定します。次のコードは、WinForms FlexGrid の行でテキストの折り返しを適用する方法を示します。

C#

```
// セルの幅に応じて、特定の行のテキストを折り返します
c1FlexGrid1.Rows[1].StyleNew.WordWrap = true;
```

VB.NET

```
' セルの幅に応じて、特定の行のテキストを折り返します
c1FlexGrid1.Rows(1).StyleNew.WordWrap = True
```

トリミングされたテキストの表示

テキストがセル幅より長い場合にテキストをトリミングして表示するには、スタイルの Trimming プロパティを設定します。このプ

ロパティは、**StringTrimming** 列挙の値を受け取ります。以下のコードを使用して、WinForms FlexGrid の行内の長いテキストをトリミングし、省略符記号を表示します。

C#

```
// 長いテキストをトリミングして、最後に省略記号を表示します
c1FlexGrid1.Rows[1].StyleNew.Trimming = StringTrimming.EllipsisCharacter;
```

VB.NET

```
' 長いテキストをトリミングして、最後に省略記号を表示します
c1FlexGrid1.Rows(1).StyleNew.Trimming = StringTrimming.EllipsisCharacter
```

テキストの配置

TextAlign プロパティを使用して、テキストの配置、つまりセルに対するテキストの位置を設定できます。このプロパティは、**TextAlignEnum** 列挙に含まれる値を受け取ります。

以下のコードを使用して、WinForms FlexGrid の行にテキストの配置を適用します。

C#

```
// テキストの配置を設定します
c1FlexGrid1.Rows[1].StyleNew.TextAlign = TextAlignEnum.LeftCenter;
```

VB.NET

```
' テキストの配置を設定します
c1FlexGrid1.Rows(1).StyleNew.TextAlign = TextAlignEnum.LeftCenter
```

テキストエフェクトの適用

テキストにさまざまなエフェクトを設定するには、**TextEffectEnum** 列挙の値を受け取る **TextEffect** プロパティを使用します。

以下のコードを使用して、WinForms FlexGrid の行にテキストを適用します。

C#

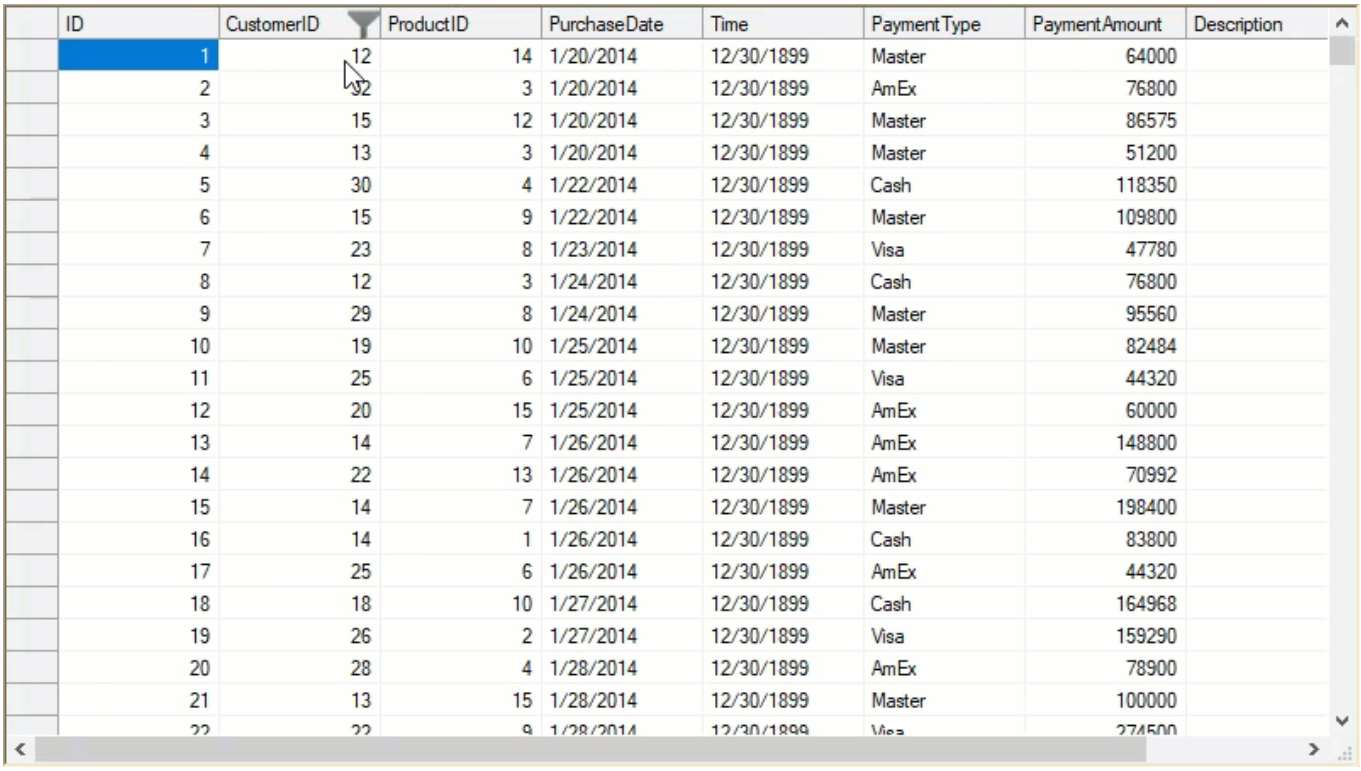
```
// 上げ表示するテキストを設定します
c1FlexGrid1.Rows[1].StyleNew.TextEffect = TextEffectEnum.Raised;
```

VB.NET

```
' テキストの配置を設定します
c1FlexGrid1.Rows(1).StyleNew.TextEffect = TextEffectEnum.Raised
```

カスタムグリフ

FlexGrid では、グリッド内で列フィルタ処理やソートなどのさまざまなアクションを示すために使用されるデフォルトのグリフ画像を変更できます。このグリッドの動作は GridGlyphs クラスを通して公開されます。このクラスは、フィルタ処理された状態、ソート順序、チェックボックスの状態などを表すためにグリッドによって使用される画像のコレクションです。これらの画像には、GlyphEnum 列挙の値を受け取る C1FlexGrid クラスの Glyphs プロパティを通してアクセスできます。



ID	CustomerID	ProductID	PurchaseDate	Time	PaymentType	PaymentAmount	Description
1	12	14	1/20/2014	12/30/1899	Master	64000	
2	32	3	1/20/2014	12/30/1899	AmEx	76800	
3	15	12	1/20/2014	12/30/1899	Master	86575	
4	13	3	1/20/2014	12/30/1899	Master	51200	
5	30	4	1/22/2014	12/30/1899	Cash	118350	
6	15	9	1/22/2014	12/30/1899	Master	109800	
7	23	8	1/23/2014	12/30/1899	Visa	47780	
8	12	3	1/24/2014	12/30/1899	Cash	76800	
9	29	8	1/24/2014	12/30/1899	Master	95560	
10	19	10	1/25/2014	12/30/1899	Master	82484	
11	25	6	1/25/2014	12/30/1899	Visa	44320	
12	20	15	1/25/2014	12/30/1899	AmEx	60000	
13	14	7	1/26/2014	12/30/1899	AmEx	148800	
14	22	13	1/26/2014	12/30/1899	AmEx	70992	
15	14	7	1/26/2014	12/30/1899	Master	198400	
16	14	1	1/26/2014	12/30/1899	Cash	83800	
17	25	6	1/26/2014	12/30/1899	AmEx	44320	
18	18	10	1/27/2014	12/30/1899	Cash	164968	
19	26	2	1/27/2014	12/30/1899	Visa	159290	
20	28	4	1/28/2014	12/30/1899	AmEx	78900	
21	13	15	1/28/2014	12/30/1899	Master	100000	
22	22	9	1/28/2014	12/30/1899	Visa	274500	

WinForms FlexGrid 内でフィルタエディタおよびフィルタ処理された列に対して使用されるデフォルトのグリフを、以下のコードを使用してカスタム画像に変更します。

C#

```
// フィルタアイコンのグリフをカスタマイズします
c1FlexGrid1.Glyphs[GlyphEnum.FilterEditor] = Image.FromFile("custom-filter-
icon.png");
// フィルタアイコンのグリフをカスタマイズします
c1FlexGrid1.Glyphs[GlyphEnum.FilteredColumn] = Image.FromFile("filter.ico");
```

VB.NET

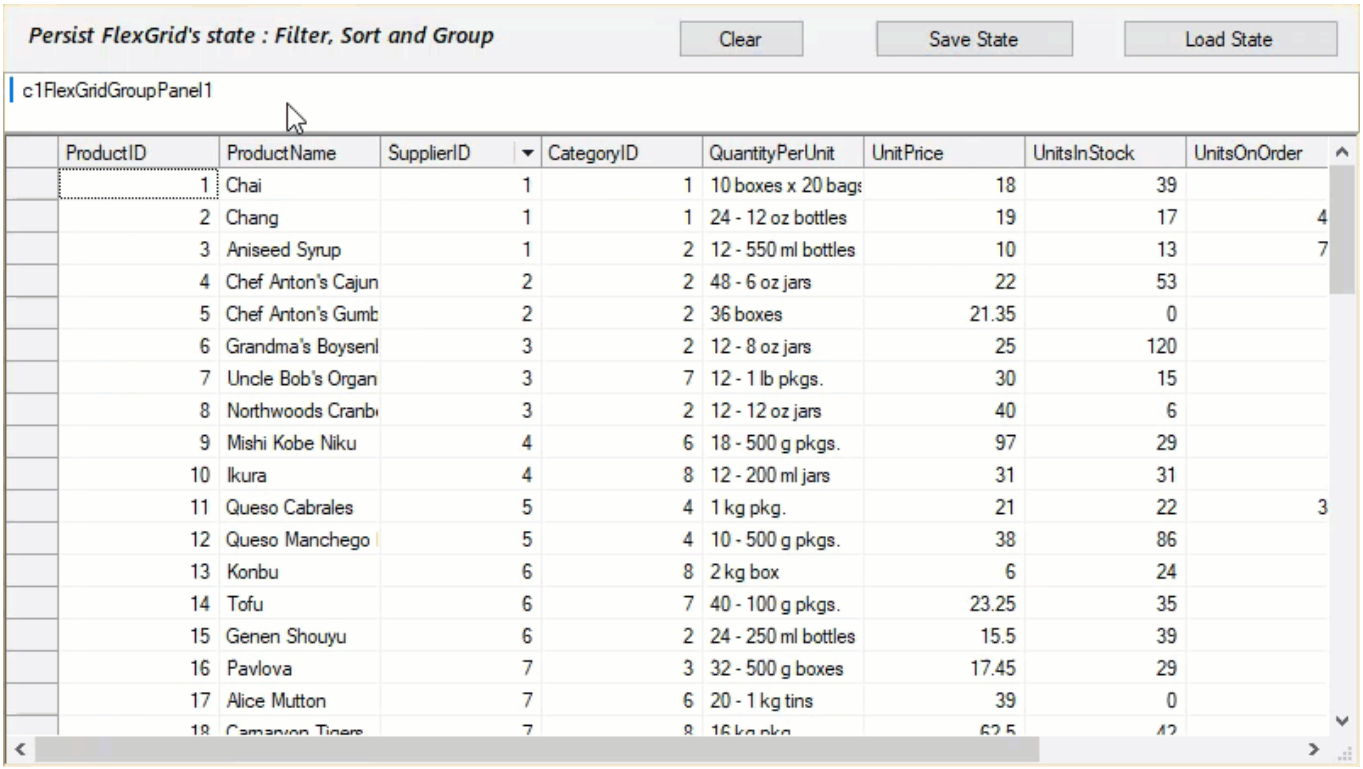
```
' フィルタアイコンのグリフをカスタマイズします
c1FlexGrid1.Glyphs(GlyphEnum.FilterEditor) = Image.FromFile("custom-filter-
icon.png")
' フィルタアイコンのグリフをカスタマイズします
c1FlexGrid1.Glyphs(GlyphEnum.FilteredColumn) = Image.FromFile("filter.ico")
```

このプロパティに null を設定すると、デフォルトのグリフに戻ります。グリフを非表示にするには、**Glyph** プロパティに空白の画像を設定してください。

状態の永続化

グリッドの永続化とは、グリッドの現在の状態を後から利用できるように保存し、必要に応じてそれを復元することです。たとえば、グループ化、ソート、フィルタ処理、スタイルなどを適用したグリッドを、さらに何らかの操作を施した後やアプリケーションを再度起動した後に使用したい場合があります。これは、グリッドの状態を永続化することで可能になります。

グリッド状態の永続化を実装する手順は 2 つです。最初にグリッド状態をシリアライズし、次にそれを復元します。このトピックでは、フィルタ処理、ソート、グループ化を行った FlexGrid の状態を永続化する方法について説明します。



グリッド状態の永続化を実装するには、System.XML 名前空間の XMLWriter および XMLReader クラスを使用します。グリッド状態を保存するには、WriteStartElement メソッドを使用して、指定された開始タグを書き込み、WriteAttributeString メソッドを使用して、フィルタ、ソート、およびグループ化の属性を書き込みます。最後に、WriteEndElement メソッドを呼び出して WriteStartElement メソッドを閉じ、ストリーム、文字列、またはファイルの形式で状態を保存します。

後で使用するために WinForms FlexGrid の状態をシリアライズして XML ファイルにするには、次のコードを使用します。

C#

```
//フィルタ、グループ、並べ替えの定義をファイルに書き込みます
private void WriteStateToXML(string filePath) {
    using (XmlWriter writer = XmlWriter.Create(filePath))
    {
        writer.WriteStartElement("FlexGrid");
        writer.WriteStartElement("Definition", "");
        writer.WriteAttributeString("Filter", flexGrid.FilterDefinition);
        writer.WriteAttributeString("Sort", flexGrid.SortDefinition);
        writer.WriteAttributeString("Group", flexGrid.GroupDefinition);
        writer.WriteEndElement();
        writer.WriteEndElement();
        writer.Flush();
    }
```

VB.NET

```
'フィルタ、グループ、並べ替えの定義をファイルに書き込みます
Private Sub WriteStateToXML(ByVal filePath As String)
```

```
    Using writer = XmlWriter.Create(filePath)
        writer.WriteStartElement("FlexGrid")
        writer.WriteStartElement("Definition", "")
        writer.WriteAttributeString("Filter", flexGrid.FilterDefinition)
        writer.WriteAttributeString("Sort", flexGrid.SortDefinition)
        writer.WriteAttributeString("Group", flexGrid.GroupDefinition)
        writer.WriteEndElement()
        writer.WriteEndElement()
        writer.Flush()
    End Using
End Sub
```

保存された状態をグリッドにロードし直すには、オブジェクトの GetAttribute メソッドを使用して、フィルタ属性、ソート属性、グループ属性を読み取ります。次に、それらを FlexGrid の FilterDefinition、SortDefinition、または GroupDefinition プロパティにそれぞれ割り当てます。

WinForms FlexGrid の保存された状態をロードするには、次のコードを使用します。

C#

```
//保存したファイルからフィルタ、並べ替え、グループの定義を読み取り、FlexGridに適用します
private void ReadStateXML(string filePath) {

    using (XmlReader reader = XmlReader.Create(filePath)) {
        reader.ReadToFollowing("Definition");
        flexGrid.SortDefinition = reader.GetAttribute("Sort");
        flexGrid.FilterDefinition = reader.GetAttribute("Filter");
        flexGrid.GroupDefinition = reader.GetAttribute("Group");
        reader.Close();
    }
```

VB.NET

```
'保存したファイルからフィルタ、並べ替え、グループの定義を読み取り、FlexGridに適用します
Private Sub ReadStateXML(ByVal filePath As String)
    Using reader = XmlReader.Create(filePath)
        reader.ReadToFollowing("Definition")
        flexGrid.SortDefinition = reader.GetAttribute("Sort")
        flexGrid.FilterDefinition = reader.GetAttribute("Filter")
        flexGrid.GroupDefinition = reader.GetAttribute("Group")
        reader.Close()
    End Using
```